KSP Reference Manual

Reference Manual written by: Nicki Marinic, Klaus Baetz
Kontakt Version: 5.0.0
Last changed: June 9, 2011

Table of Contents

Callbacks

on async_complete

```
on async_complete
```

synchronization callback that is triggered after each finished operation of one of the following commands:

- `load_array()`
- `save_array()`
- `load_midi_file()`
- `save_midi_file()`
- `load_ir_sample()`

Remarks

To resolve synchronization issues, the commands mentioned above now return unique IDs when being used. Upon completion of the command's action, the `on async_complete` callback gets triggered and the built-in variable `$NI_ASYNC_ID` is updated with the ID of the command that triggered the callback. If the command was completed successfully (for example if the file was found and successfully loaded), the internal value `$NI_ASYNC_EXIT_STATUS` is set to **1**, otherwise it is **0**.

Examples

```
on init
  declare $load_midi_file_id
  declare ui_button $load_midi_file
end on

on ui_control ($load_midi_file)
  $load_midi_file_id := load_midi_file("midifile.mid")
  while ($load_midi_file_id # -1)
    wait (1)
  end while
  message ("MIDI file loaded")
end on

on async_complete
  if ($NI_ASYNC_ID = $load_midi_file_id)
    $load midi file id := -1
  end if
end on
```

a workflow example

See Also

```
$NI_ASYNC_EXIT_STATUS
$NI_ASYNC_ID
load_array()
load_ir_sample()
load_midi_file()
save_array()
save_midi_file()
```

on controller

on controller
controller callback, executed whenever a CC, pitch bend or channel pressure message is received

Examples

```
on controller
  if (in_range($CC_NUM,0,127))
        message("CC Number: "& $CC_NUM&" - Value: " & %CC[$CC_NUM])
  else
        if ($CC_NUM = $VCC_PITCH_BEND)
              message("Pitchbend" & " - Value: " & %CC[$CC_NUM])
        end if
        if ($CC_NUM = $VCC_MONO_AT)
              message("Channel Pressure" &" - Value: "&%CC[$CC_NUM])
        end if
  end if
end on
```

query CC, pitch bend and channel pressure data

See Also

set_controller()
ignore_controller
%CC[]
$CC_NUM
$VCC_PITCH_BEND
$VCC_MONO_AT

on init

```
on init
```

init callback, executed when the script was successfully analyzed

Examples

```
on init
  declare ui_button $Sync

  declare ui_menu $time
  add_menu_item ($time,"16th",0)
  add_menu_item ($time,"8th",1)

  $Sync := 0
  {sync is off by default, so hide menu}
  move_control ($time,0,0)
  move_control ($Sync,1,1)

  make_persistent ($Sync)
  make_persistent ($time)

  read_persistent_var ($Sync)
  if ($Sync = 1)
        move_control ($time,2,1)
  else
        move_control ($time,0,0)
  end if
end on

on ui_control ($Sync)
  if ($Sync = 1)
        move_control ($time,2,1)
  else
        move_control ($time,0,0)
  end if
end on
```

init callback with read_persistent_var()

See Also

make_persistent
read_persistent_var

on listener

```
on listener
```

listener callback, executed whenever a host's transport command is received or at definable time intervals

Remarks

- This is the callback for most things that are somehow time based or need to be synced to a host software's master tempo.
- Triggering the on listener callback more frequently increases the CPU load.
- It is very important to know that the on listener callback works in real time and therefore it is crucial to be careful when working with it. It is possible that in some special situations (like tempo changes within the host), ticks are left out. As an example, a modulo division to receive the trigger for a special timing (like offbeats) could return wrong results. So you always should use a certain tolerance and re-check every now in then that you are still in sync and that you still get the results you would expect from your script.
- KONTAKT'S stand-alone mode always uses a 4/4 time signature.

Examples

```
on init
  declare $mscount
  declare $beatcount

  set_listener ($NI_SIGNAL_TRANSP_STOP,1)
  set_listener ($NI_SIGNAL_TRANSP_START,1)
  set_listener ($NI_SIGNAL_TIMER_MS,1000000)
  set_listener ($NI_SIGNAL_TIMER_BEAT,1)
end on

on listener
  select ($NI_SIGNAL_TYPE)
        case $NI_SIGNAL_TRANSP_STOP
              message ("Playback was stopped.")
        case $NI_SIGNAL_TRANSP_START
              message ("Playback was started.")
        case $NI_SIGNAL_TIMER_MS
              inc ($mscount)
              message ($mscount & "sec. / " & $beatcount & " beats")
        case $NI_SIGNAL_TIMER_BEAT
              inc ($beatcount)
              message ($mscount & "sec. / " & $beatcount & " beats")
  end select
end on
```

a timer and a bpm counter

See Also

```
set_listener()
change_listener_par()
$NI_SIGNAL_TYPE
$NI_SONG_POSITION
```

on note

```
on note
```

note callback, executed whenever a note on message is received

Examples

```
on note
  message ("Note Number: " & $EVENT_NOTE ...
  & " - Velocity: " & $EVENT_VELOCITY)
end on
```

query note data

See Also

on release
ignore_event()

on pgs_changed

```
on pgs_changed
```

callback type, executed whenever any `pgs_set_key_val()` command is executed in any script

Remarks

pgs stands for Program Global Storage and is a means of communication between script slots. See the chapter on PGS for more details.

Examples

```
on init
  pgs_create_key(FIRST_KEY, 1) {defines a key with 1 element}
  pgs_create_key(NEXT_KEY, 128){defines a key with 128 elements}
 declare ui_button $Push
end on

on ui_control($Push)
  pgs_set_key_val(FIRST_KEY, 0,70 * $Push)

  pgs_set_key_val(NEXT_KEY, 0, 50 * $Push)
  pgs_set_key_val(NEXT_KEY, 127, 60 * $Push)
end on
```

Example 1 – pressing the button...

```
on init
  declare ui_knob $First (0,100,1)
  declare ui_table %Next[128] (5,2,100)
end on

on pgs_changed

{checks if FIRST_KEY and NEXT_KEY have been declared}
  if(pgs_key_exists(FIRST_KEY) and ...
     pgs_key_exists(NEXT_KEY))
        $First := pgs_get_key_val(FIRST_KEY,0)
        %Next[0] := pgs_get_key_val(NEXT_KEY,0)
        %Next[127] := pgs_get_key_val(NEXT_KEY,127)
  end if
end on
```

will change the controls in this example, regardless of the script slot order.

See Also

PGS

on poly_at

```
on poly_at
```

polyphonic aftertouch callback, executed whenever a polyphonic aftertouch message is received

Examples

```
on init
  declare %note_id[128]
end on

on note
  %note_id[$EVENT_NOTE] := $EVENT_ID
end on

on poly_at
  change_tune(%note_id[$POLY_AT_NUM],%POLY_AT[$POLY_AT_NUM]*1000,0)
end on
```

a simple poly aftertouch to pitch implementation

See Also

%POLY_AT[]
$POLY_AT_NUM
$VCC_MONO_AT

on release

```
on release
```

release callback, executed whenever a note off message is received

Examples

```
on init
  declare $new_id
end on

on release
  wait(1000)
  $new_id := play_note($EVENT_NOTE,$EVENT_VELOCITY,0,100000)
  change_vol ($new_id,-24000,1)
end on
```

creating an artificial release noise

See Also

on note
ignore_event()

on rpn/nrpn

```
on rpn/nrpn
```

rpn and nrpn callbacks, executed whenever a rpn or nrpn (registered/nonregistered parameter number) message is received

Examples

```
on rpn
  select ($RPN_ADDRESS)
        case 0
        message ("Pitch Bend Sensitivity"&" - Value: "& $RPN_VALUE)
        case 1
        message ("Fine Tuning" & " - Value: " & $RPN_VALUE)
        case 2
        message ("Coarse Tuning" & " - Value: " & $RPN_VALUE)
  end select
end on
```

query standard rpn messages

See Also

on controller
set_rpn/set_nrpn
msb()/lsb()
$RPN_ADDRESS
$RPN_VALUE

on ui_control()

```
on ui_control(<variable>)
```

UI callback, executed whenever the user changes the respective UI element

Examples

```
on init
  declare ui_knob $Knob (0,100,1)
  declare ui_button $Button
  declare ui_switch $Switch
  declare ui_table %Table[10] (2,2,100)
  declare ui_menu $Menu
  add_menu_item ($Menu,"Entry 1",0)
  add_menu_item ($Menu,"Entry 2",1)
  declare ui_value_edit $VEdit (0,127,1)
  declare ui_slider $Slider (0,100)
end on
on ui_control ($Knob)
  message("Knob" & " (" & $ENGINE_UPTIME & ")")
end on
on ui_control ($Button)
  message("Button" & " (" & $ENGINE_UPTIME & ")")
end on
on ui_control ($Switch)
  message("Switch" & " (" & $ENGINE_UPTIME & ")")
end on
on ui_control (%Table)
  message("Table" & " (" & $ENGINE_UPTIME & ")")
end on
on ui_control ($Menu)
  message("Menu" & " (" & $ENGINE_UPTIME & ")")
end on
on ui_control ($VEdit)
  message("Value Edit" & " (" & $ENGINE_UPTIME & ")")
end on
on ui_control ($Slider)
  message("Slider" & " (" & $ENGINE UPTIME & ")")
end on
```

various ui controls and their corresponding callbacks

See Also

on ui_update

on ui_update

on ui_update
UI update callback, executed with every GUI change in Kontakt

Remarks

`on ui_update` should be used with caution, since it is triggered with every GUI change in Kontakt.

Examples

```
on init
  declare ui_knob $Volume (0,1000000,1)
  set_knob_unit ($Volume,$KNOB_UNIT_DB)
  set_knob_defval ($Volume,630859)
  $Volume := _get_engine_par ($ENGINE_PAR_VOLUME,-1,-1,-1)
  set_knob_label ($Volume,_get_engine_par_disp...
  ($ENGINE_PAR_VOLUME,-1,-1,-1))
end on

on ui_update
  $Volume := _get_engine_par ($ENGINE_PAR_VOLUME,-1,-1,-1)
  set_knob_label($Volume,_get_engine_par_disp...
  ($ENGINE_PAR_VOLUME,-1,-1,-1))
end on

on ui_control ($Volume)
  _set_engine_par($ENGINE_PAR_VOLUME,$Volume,-1,-1,-1)
  set_knob_label ($Volume,_get_engine_par_disp...
  ($ENGINE_PAR_VOLUME,-1,-1,-1))
end on
```

mirroring instrument volume with a KSP control

See Also

`on ui_control()`

Variables

$ (variable)

```
declare $<variable-name>
```

declare a user-defined normal variable to store a single integer value

Remarks

- Please do not create variables with the prefixes $NI_, $CONTROL_PAR_, $EVENT_PAR_ or $ENGINE_PAR_ as these prefixes are used for internal variables and constants.

Examples

```
on init
  declare $testvariable
  $testvariable := 123
  $testvariable := $testvariable + 456
  message ($testvariable)
end on
```

query CC, pitch bend and channel pressure data

See Also

`on init`

const $ (constant)

```
declare const $<variable-name>
```

declare a user-defined constant variable to store a single integer value

Remarks

- Please do not create constants with the prefixes $NI_, $CONTROL_PAR_, $EVENT_PAR_ or $ENGINE_PAR_ as these prefixes are used for internal variables and constants.

Examples

```
on init
  declare const $testconstant := 123
  message ($testconstant)
end on
```

query CC, pitch bend and channel pressure data

See Also

`on init`

polyphonic $ (polyphonic variable)

```
declare polyphonic $<variable-name>
```

declare a user-defined polyphonic variable to store a single integer value.

Remarks

- A polyphonic variable acts as a unique variable for each executed event, avoiding conflicts in callbacks that are executed in parallel.
- A polyphonic variable retains its value in the release callback of the corresponding note.
- Polyphonic variables need much more memory than normal variables.
- Please do not create variables with the prefixes $NI_, $CONTROL_PAR_, $EVENT_PAR_ or $ENGINE_PAR_ as these prefixes are used for internal variables and constants.

Examples

```
on init
  declare polyphonic $a
  {declare $a}
end on

on note
  ignore_event($EVENT_ID)

  $a:= 0
  while ($a < 13 and $NOTE_HELD = 1)
  play_note($EVENT_NOTE+$a,$EVENT_VELOCITY,0,$DURATION_QUARTER/2)
        inc($a)
        wait($DURATION_QUARTER)
  end while
end on
```

to hear the effect of the polyphonic variable, play and hold an octave: both notes will ascend chromatically. Then make $a a normal variable and play the octave again: $a will be shared by both executed callbacks, thus both notes will ascend in larger intervals

See Also

`wait()`

% (array)

```
declare %<array-name>[<num-of-elements>]
```

declare a user-defined array to store single integer values at specific indices

Remarks

- The maximal size of arrays is 32768.
- Arrays have to be declared with a constant value.
- Please do not create arrays with the prefixes %NI_, %CONTROL_PAR_, %EVENT_PAR_ or %ENGINE_PAR_ as these prefixes are used for internal variables and constants.

Examples

```
on init
  declare %presets[10*8] := (...
  {1}   8,8,8,0,  0,0,0,0,...
  {2}   8,8,8,8,  0,0,0,0,...
  {3}   8,8,8,8,  8,8,8,8,...
  {4}   0,0,5,3,  2,0,0,0,...
  {5}   0,0,4,4,  3,2,0,0,...
  {6}   0,0,8,7,  4,0,0,0,...
  {7}   0,0,4,5,  4,4,2,2,...
  {8}   0,0,5,4,  0,3,0,0,...
  {9}   0,0,4,6,  7,5,3,0,...
  {10}  0,0,5,6,  4,4,3,2)
end on
```

creating an array for storing preset data

See Also

Array and Group Commands

@ (string variable)

```
declare @<variable-name>
```

declare a user-defined string variable to store text

Examples

```
on init
  declare @text
  @text := "Last received note number played or released: "
end on

on note
  message(@text & $EVENT_NOTE)
end on
on release
  message(@text & $EVENT_NOTE)
end on
```

use string variables to display long text

See Also

`! (string array)`

! (string array)

```
declare !<array-name>
```

declare a user-defined string array to store text strings at specified indices

Examples

```
on init
  declare $count

  declare !note[12]
  !note[0] := "C"
  !note[1] := "Db"
  !note[2] := "D"
  !note[3] := "Eb"
  !note[4] := "E"
  !note[5] := "F"
  !note[6] := "Gb"
  !note[7] := "G"
  !note[8] := "Ab"
  !note[9] := "A"
  !note[10] := "Bb"
  !note[11] := "B"

  declare !name [128]
  while ($count < 128)
        !name[$count] := !note[$count mod 12] & (($count/12)-2)
        inc ($count)
  end while
end on

on note
  message("Note played: " & !name[$EVENT_NOTE])
end on
```

creating a string array with all MIDI note names

See Also

`@ (string variable)`

Control Statements

if…else…end if

if…else…end if
declare a user-defined normal variable to store a single integer value

Examples

```
on controller
  if (in_range($CC_NUM,0,127))
        message("CC Number: "& $CC_NUM&" - Value: " & %CC[$CC_NUM])
  else
        if ($CC_NUM = $VCC_PITCH_BEND)
              message("Pitchbend" & " - Value: " & %CC[$CC_NUM])
        end if
        if ($CC_NUM = $VCC_MONO_AT)
              message("Channel Pressure" &" - Value: "&%CC[$CC_NUM])
        end if
  end if
end on
```

query CC, pitch bend and channel pressure data

See Also

`select()`

select()

select(<variable>)…end select
select statement

Remarks

The `select` statement is similar to the `if` statement, except that it has an arbitrary number of branches. The expression after the select keyword is evaluated and matched against the single case branches, the first case branch that matches is executed.
The case branches may consist of either a single constant number or a number range (expressed by the term "`x to y`").

Examples

```
on controller
  if ($CC_NUM = $VCC_PITCH_BEND)
        select (%CC[$VCC_PITCH_BEND])
              case -8192 to -1
                    message("Pitch Bend down")
              case 0
                    message("Pitch Bend center")
              case 1 to 8191
                    message("Pitch Bend up")
        end select
  end if
end on
```

query the state of the pitch bend wheel

See Also

`if_else_end if`

while()

```
while(<condition>)…end while
```

while loop

Examples

```
on note
  ignore_event($EVENT_ID)
  while($NOTE_HELD = 1)
  play_note($EVENT_NOTE,$EVENT_VELOCITY,0,$DURATION_QUARTER/2)
        wait($DURATION_QUARTER)
  end while
end on
```

repeating held notes at the rate of one quarter note

See Also

$NOTE_HELD

Operators

Boolean Operators

Boolean operators are used in `if` and `while` statements, since they return if the condition is either true or false. Below is a list of all Boolean operators. x, y and z denote numerals, a and b stand for Boolean values.

Boolean Operators	
`x > y`	greater than
`x > y`	less than
`x >= y`	greater than or equal
`x <= y`	less than or equal
`x = y`	equal
`x # y`	not equal
`in_range(x,y,z)`	true if x is between y and z
`not a`	true if a is false and vice versa
`a and b`	true if a is true and b is true
`a or b`	true if a is true or b is true

Arithmetic Operators

The following arithmetic operators can be used:

Arithmetic operators	
`x + y`	addition
`x - y`	subtraction
`x * y`	multiplication
`x / y`	division
`x mod y`	modulo
`-x`	negative value

Bit Operators

The following bit operators can be used:

Bit Operators	
x .and. y x .or. y .not. x	bitwise and bitwise or bitwise negation
sh_left(<expression>,<shift-bits>)	shifts the bits in <expression> by the amount of <shift-bits> to the left
sh_right(<expression>,<shift-bits>)	shifts the bits in <expression> by the amount of <shift-bits> to the right

See Also

abs()
dec()
inc()
random()

Array and Group Commands

array_equal()

```
array_equal(<array1-variable>,<array2-variable>)
```

check the values of two arrays, true if all values are equal, false if not

Examples

```
on init
  declare %array_1[10]
  declare %array_2[11]

  if (array_equal(%array_1,%array_2))
        message($ENGINE_UPTIME)
  end if

end on
```

this script will produce an error message since the the two arrays don't have the same size

See Also

sort()
num_elements()
search()

num_elements()

```
num_elements(<array-variable>)
```

return the number of elements in an array

Remarks

With this function you can, e.g., check how many groups are affected by the current event by using `num_elements(%GROUPS_AFFECTED)`.

Examples

```
on note
  message(num_elements(%GROUPS_AFFECTED))
end on
```

outputs the number of groups playing

See Also

`array_equal()`
`sort()`
`search()`
`%GROUPS_AFFECTED`

search()

```
search(<array-variable>,<value>)
```

searches the specified array for the specified value and returns the index of its first position.
If the value is not found, the function returns -1

Examples

```
on init
  declare ui_table %array[10] (2,2,5)
  declare ui_button $check
  set_text ($check,"Zero present?")
end on

on ui_control ($check)
  if (search(%array,0) = -1)
       message ("No")
   else
       message("Yes")
   end if
   $check := 0
end on
```

checking if a specific value is present

See Also

array_equal()
num_elements()
sort()

sort()

```
sort(<array-variable>,<direction>)
```

sort an array in ascending or descending order.

With direction = 0, the array is sorted in **ascending** order
With direction # 0, the array is sorted in **descending** order

Examples

```
on init
  declare $count
  declare ui_table %array[128] (3,3,127)

  while ($count < 128)
        %array[$count] := $count
        inc($count)
  end while
  declare ui_button $Invert

end on

on ui_control ($Invert)
  sort(%array,$Invert)
end on
```

quickly inverting a linear curve display

See Also

array_equal()
num_elements()
sort()

allow_group()

```
allow_group(<group-index>)
```

turn on the specified group,i.e. make it available for playback

Remarks

- Note that the numbering of the group index is zero based, i.e. the first group has the group index 0.
- The groups can only be changed if the voice is not running.

Examples

```
on note
  disallow_group($ALL_GROUPS)
  allow_group(0)
end on
```

only the first group will play back

See Also

```
$ALL_GROUPS
$EVENT_PAR_ALLOW_GROUP
disallow_group()
```

disallow_group()

```
disallow_group(<group-index>)
```

turn off the specified group,i.e. make it unavailable for playback

Remarks

- Note that the numbering of the group index is zero based, i.e. the first group has the group index 0.
- The groups can only be changed if the voice is not running.

Examples

```
on init
  declare $count
  declare ui_menu $groups_menu

  add_menu_item ($groups_menu,"Play All",-1)
  while ($count < $NUM_GROUPS)
        add_menu_item ($groups_menu,"Mute: " & ...
        group_name($count),$count)
        inc($count)
  end while
end on
on note
  if ($groups_menu # -1)
        disallow_group($groups_menu)
  end if
end on
```

muting one specific group of an instrument

See Also

$ALL_GROUPS
$EVENT_PAR_ALLOW_GROUP
allow_group()

find_group()

```
find_group(<group-name>)
```

returns the group index for the specified group name

Examples

```
on note
  disallow_group(find_group("Accordion"))
end on
```

a simple, yet useful script

See Also

allow_group()
disallow_group
group_name()

get_purge_state()

```
get_purge_state(<group-index>)
```

checks if the specified group is purged (**0**), or not (**1**)

`<group-index>`	the index number of the group that should be checked

Examples

```
on init
  declare ui_button $purge
  declare ui_button $checkpurge
  set_text ($purge,"Purge 1st Group")
  set_text ($checkpurge,"Check purge status")
end on

on ui_control ($purge)
  purge_group(0,abs($purge-1))
end on

on ui_control ($checkpurge)
  if (get_purge_state(0) = 0)
        message(“Group is purged.”)
  else
        message(“Group is not purged.”)
  end if
end on
```

a simple purge check

See Also

`purge_group`

group_name()

```
group_name(<group-index>)
```

returns the group name for the specified group

Remarks

Note that the numbering of the group index is zero based, i.e. the first group has the group index 0.

Examples

```
on init
  declare $count
  declare ui_menu $groups_menu

  while ($count < $NUM_GROUPS)
        add_menu_item ($groups_menu,group_name($count),$count)
        inc($count)
  end while
end on
```

quickly creating a menu with all available groups

```
on init
  declare $count
  declare ui_label $label (2,6)
  set_text($label,"")
end on
on note
  $count := 0
  while ($count < num_elements(%GROUPS_AFFECTED))
        add_text_line($label,group_name(%GROUPS_AFFECTED[$count]))
        inc($count)
  end while
end on
on release
  set_text($label,"")
end on
```

display the names of the sounding groups

See Also

$ALL_GROUPS
$NUM_GROUPS
allow_group()
disallow_group
find_group()
output_channel_name()

load_array()

load_array(<array>,<mode>)	
loads an array from an external file.	
<array> <mode>	the name of the array that is to be loaded. **0:** A dialog opens up that allows you to load one array. Can only be used in an ui_callback and in an pgs_callback. **1:** The array is directly loaded from the /Data folder besides the resource container. In addition to the ui and pgs callbacks, this can also be used within the init callback.

Remarks

- Since KONTAKT 5, it is also possible to save and load string arrays.
- Be aware that the name of the array will also be saved so you can't load the saved array %xyz to array %abc.
- It is very important to know that the array data is not directly available after the `load_array` command has been executed. The callback continues but it takes a short time until you can work with the loaded array. The only situation in which the values are instantly available is when using mode 1 within an init callback. To solve this problem, please make use of the `on async_complete` callback.
- This command returns a unique value after is has finished its action. Please use it in combination with `$NI_ASYNC_ID` within the `on_async_complete` callback to avoid synchronization issues.
- You should also take note that when using mode 0 the callback continues even if the loading dialog is still open.
- When loading an array within the init callback, please remember that the loaded data will be overwritten at the end of the callback if the array is persistent. Use `read_persistent_var` before loading the array to avoid this problem.

Example

```
on init
  declare %save[1]
  declare $sync_id
  declare ui_slider $Slider (0,100)
  declare ui_menu $Menu
  add_menu_item($Menu,"Save",0)
  add_menu_item($Menu,"Load",1)
end on

on async_complete
  if ($NI_ASYNC_ID = $sync_id)
        $sync_id := -1
  end if
end on

on ui_control ($Menu)
  select ($Menu)
        case 0
              save_array (%save,0)
        case 1
              $sync_id := load_array (%save,0)
              while ($sync_id # -1)
                    wait (1)
              end while
              $Slider := %save[0]
  end select
end on

on ui_control($Slider)
  %save[0] := $Slider
end on
```

Saving and loading the value of a slider.

See Also

$NI_ASYNC_ID
on async_complete
save_array()

purge_group()

purge_group(<group-index>,<mode>)	
purges (i.e. unload) the samples of the specified group	
<group-index>	the index number of the group which contains the samples to be purged
<mode>	If set to **0**, the samples of the specified group are unloaded. If set to **1**, the samples are reloaded.

Remarks

- `purge_group()` is an advanced command, so it should be used with caution. When using `purge_group()` in a while loop, don't use any wait commands within the loop.
- `purge_group()` can only be used in an ui callback

Examples

```
on init
  declare ui_button $purge
  set_text ($purge,"Purge 1st Group")
end on

on ui_control ($purge)
  purge_group(0,abs($purge-1))
end on
```

unloading all samples of the first group

See Also

`get_purge_state`

save_array()

save_array(<array>,<mode>)	
saves an array into an external file.	
<array> <mode>	the name of the array that is to be saved. **0:** A dialog opens up that allows you to save one array. Can only be used in an ui_callback and in a pgs_callback. **1:** The array is directly saved into the /Data folder besides the resource container. In addition to the ui and pgs callbacks, this can also be used within the init callback.

Remarks

- Since KONTAKT 5, it is also possible to save and load string arrays.
- Be aware that the name of the array will also be saved so you can't load the saved array %xyz to array %abc.
- You should also take note that when using mode 0 the callback continues even if the save dialog is still open.
- This command returns a unique value after is has finished its action. Please use it in combination with `$NI_ASYNC_ID` within the `on_async_complete` callback to avoid synchronization issues.

Examples

```
on init
  declare %save[1]
  declare $sync_id
  declare ui_slider $Slider (0,100)
  declare ui_menu $Menu
  add_menu_item($Menu,"Save",0)
  add_menu_item($Menu,"Load",1)
end on

on async_complete
  if ($NI_ASYNC_ID = $sync_id)
        $sync_id := -1
  end if
end on

on ui_control ($Menu)
  select ($Menu)
        case 0
              save_array (%save,0)
        case 1
              $sync_id := load_array (%save,0)
              while ($sync_id # -1)
                    wait (1)
              end while
              $Slider := %save[0]
  end select
end on

on ui_control($Slider)
  %save[0] := $Slider
end on
```

Saving and loading the value of a slider.

See Also

$NI_ASYNC_ID
load_array()
on async_complete

User Interface Commands

add_menu_item()

add_menu_item(<variable>,<text>,<value>)	
create a menu entry	
<variable>	the variable of the ui menu
<text>	the text of the menu entry
<value>	the value of the menu entry

Remarks

You can only create menu entries in the init callback but you can change their <text> and <value> afterwards by using set_menu_item_str and set_menu_item_value.
Add as many menu entries as you would possibly need within the init callback and then show or hide them dynamically by using set_menu_item_visibility.

Examples

```
on init
  declare ui_menu $menu
  add_menu_item ($menu, "First Entry",0)
  add_menu_item ($menu, "Second Entry",1)
  add_menu_item ($menu, "Third Entry",2)
end on
```
a simple menu

```
on init
  declare ui_menu $menu
  add_menu_item ($menu, "Third Entry",2)
  add_menu_item ($menu, "Second Entry",1)
  add_menu_item ($menu, "First Entry",0)
end on
```
the values need not be in order

See Also

$CONTROL_PAR_SELECTED_ITEM_IDX
$CONTROL_PAR_NUM_ITEMS
get_menu_item_str()
get_menu_item_value()
get_menu_item_visibility()
set_menu_item_str()
set_menu_item_visibility()
ui_menu

add_text_line()

add_text_line (<variable>,<text>)	
add a new text line in the specified label without erasing existing text	
<variable>	the variable of the ui label
<text>	the text to be displayed

Examples

```
on init
  declare ui_label $label (1,4)
  set_text($label,"")
  declare $count
end on

on note
  inc($count)
  select ($count)
        case 1
              set_text($label, $count & ": " & $EVENT_NOTE)
        case 2 to 4
              add_text_line($label, $count & ": " & $EVENT_NOTE)
  end select
  if ($count = 4)
        $count := 0
  end if
end on
```

monitoring the last four played notes

See Also

set_text()
ui_label

attach_level_meter()

```
attach_level_meter
(<uiID>,<groupIdx>,<slotIdx>,<channelIdx>,<busIdx>)
```

attach a level meter to a certain position within the instrument to read volume data

Remarks

- For the busses the `<groupIdx>` and `<slotIdx>` have to be -1.
- The instrument volume has the following syntax:
  `attach_level_meter (<uiID>,-1,-1,<channelIdx>,-1)`

Examples

```
on init
  declare ui_level_meter $Level1
  declare ui_level_meter $Level2
  attach_level_meter (get_ui_id($Level1),-1,-1,0,-1)
  attach_level_meter (get_ui_id($Level2),-1,-1,1,-1)
end on
```

creating two volume meters, each one displaying one side of Kontakt's instrument output

See Also

$CONTROL_PAR_BG_COLOR
$CONTROL_PAR_OFF_COLOR
$CONTROL_PAR_ON_COLOR
$CONTROL_PAR_OVERLOAD_COLOR
$CONTROL_PAR_PEAK_COLOR
$CONTROL_PAR_VERTICAL
ui_level_meter

attach_zone()

attach_zone (<variable>,<zone_id>,<flags>)	
connects the corresponding zone to the waveform so that it shows up within the display	
<variable>	the variable of the ui waveform
<zone_id>	the ID of the zone that you want to attach to the waveform
<flags>	you can control different settings of the UI waveform via its flags. The following flags are available: $UI_WAVEFORM_USE_SLICES $UI_WAVEFORM_USE_TABLE $UI_WAVEFORM_TABLE_IS_BIPOLAR $UI_WAVEFORM_USE_MIDI_DRAG

Examples

```
on init
  declare ui_waveform $Waveform(6,6)
  attach_zone ($Waveform,find_zone(”Test”),$UI_WAVEFORM_USE_SLICES)
end on
```

attaches the zone “Test” to the waveform and displays the zone’s slices

See Also

set_ui_wf_property()
get_ui_wf_property()
ui_waveform()
find_zone()
Waveform Flag Constants
Waveform Property Constants

hide_part()

hide_part(<variable>,<hide-mask>)	
hide specific parts of user interface controls	
<variable>	the name of the ui control
<hide-mask>	bit by bit combination of the following constants: $HIDE_PART_BG {Background of knobs, labels, value edits and tables} $HIDE_PART_VALUE {value of knobs} $HIDE_PART_TITLE {title of knobs} $HIDE_PART_MOD_LIGHT {mod ring light of knobs}

Remarks

hide_part() is only available in the init callback.

Examples

```
on init
  declare ui_knob $Knob (0,100,1)

  hide_part($Knob,$HIDE_PART_BG...
  .or. $HIDE_PART_MOD_LIGHT...
  .or. $HIDE_PART_TITLE...
  .or. $HIDE_PART_VALUE)

end on
```

a naked knob

```
on init
  declare ui_label $label_1 (1,1)
  set_text ($label_1,"Small Label")
  hide_part ($label_1,$HIDE_PART_BG)
end on
```

hide the background of a label (also possible with other ui elements)

See Also

$CONTROL_PAR_HIDE
$HIDE_PART_NOTHING
$HIDE_WHOLE_CONTROL

fs_get_filename()

fs_get_filename(<ui-ID>,<return-parameter>)	
return the filename of the last selected file in the UI file browser.	
<ui-ID>	the ID number of the ui control. You can retrieve the ID number with get_ui_id()
<return-parameter>	**0:** returns the filename without extension **1:** returns the filename with extension **2:** returns the whole path

Remarks

All paths for saving or loading anything as well as setting any kind of path for the ui_file_selector element should be formatted in the following way: **/C:/myFolder/mysubfolder**
It is necessary to use a "/" (slash character) and a "\" (backslash character) as a folder separator, even for Windows based systems. In addition to that, the full path has to start with a "/". All paths returned by the get_folder() and the fs_get_filename() commands are in this format.

Examples

```
on init
  declare $MIDISyncID
  declare ui_file_selector $fsMIDISelector
  set_control_par_str(get_ui_id($fsMIDISelector),...
  $CONTROL_PAR_BASEPATH,"/C:/NI Testpath")
  set_control_par_str(get_ui_id($fsMIDISelector),...
  $CONTROL_PAR_FILEPATH,"/C:/NI Testpath")
  set_control_par(get_ui_id($fsMIDISelector),...
  $CONTROL_PAR_FILE_TYPE,$NI_FILE_TYPE_MIDI)
  set_control_par(get_ui_id($fsMIDISelector),...
  $CONTROL_PAR_COLUMN_WIDTH,100)
  set_control_par(get_ui_id($fsMIDISelector),...
  $CONTROL_PAR_HEIGHT,100)
  set_control_par(get_ui_id($fsMIDISelector),...
  $CONTROL_PAR_WIDTH,550)
end on

on async_complete
  if ($NI_ASYNC_ID = $MIDISyncID)
        $MIDISyncID := -1
  end if
end on

on ui_control ($fsMIDISelector)
  $MIDISyncID := load_midi_file...
  (fs_get_filename(get_ui_id($fsMIDISelector),2))
  while ($MIDISyncID # -1)
        wait (100)
  end while
  message ("MIDI File loaded")
end on
```

loading MIDI files via ui file selector

See Also

```
fs_navigate()
ui_file_selector
```

fs_navigate()

fs_navigate(<ui-ID>,<direction>)	
jump to the next/previous file in an ui file selector and trigger its callback.	
<ui-ID>	the ID number of the ui control. You can retrieve the ID number with get_ui_id()
<direction>	**0:** the previous file (in relation to the currently selected one) is selected **1:** the next file (in relation to the currently selected one) is selected

Examples

```
on init
  declare $MIDISyncID
  declare ui_file_selector $fsMIDISelector
  set_control_par_str(get_ui_id($fsMIDISelector),...
  $CONTROL_PAR_BASEPATH,"/C:/NI Testpath")
  set_control_par_str(get_ui_id($fsMIDISelector),...
  $CONTROL_PAR_FILEPATH,"/C:/NI Testpath")
  set_control_par(get_ui_id($fsMIDISelector),...
  $CONTROL_PAR_FILE_TYPE,$NI_FILE_TYPE_MIDI)
  set_control_par(get_ui_id($fsMIDISelector),...
  $CONTROL_PAR_COLUMN_WIDTH,100)
  set_control_par(get_ui_id($fsMIDISelector),...
  $CONTROL_PAR_HEIGHT,100)
  set_control_par(get_ui_id($fsMIDISelector),...
  $CONTROL_PAR_WIDTH,550)
  declare ui_button $prev
  declare ui_button $next
  move_control ($prev,1,6)
  move_control ($next,2,6)
end on

on async_complete
  if ($NI_ASYNC_ID = $MIDISyncID)
        $MIDISyncID := -1
  end if
end on

on ui_control ($fsMIDISelector)
  $MIDISyncID := load_midi_file...
  (fs_get_filename(get_ui_id($fsMIDISelector),2))
  while ($MIDISyncID # -1)
        wait (100)
  end while
  message ("MIDI File loaded")
end on

on ui_control ($prev)
  fs_navigate (get_ui_id($fsMIDISelector),0)
end on

on ui_control ($next)
  fs_navigate (get_ui_id($fsMIDISelector),1)
end on
```

creating two navigation buttons to make browsing easier

See Also

```
fs_get_filename()
ui_file_selector
```

get_control_par()

get_control_par(<ui-ID>,<control-parameter>)	
retrieve various parameters of the specified gui control	
<ui-ID>	the ID number of the ui control. You can retrieve the ID number with get_ui_id()
<control-parameter>	the control parameter variable like $CONTROL_PAR_WIDTH

Remarks

get_control_par() comes in two additional flavors, get_control_par_str() for the usage with text strings and get_control_arr() for working with arrays.

Examples

```
on init
  declare ui_value_edit $Test (0,100,1)
  message(get_control_par(get_ui_id($Test),...
  $CONTROL_PAR_WIDTH))
end on
```

retrieving the width of a value edit in pixels

See Also

set_control_par()
$CONTROL_PAR_KEY_SHIFT
$CONTROL_PAR_KEY_ALT
$CONTROL_PAR_KEY_CONTROL

get_menu_item_str()

```
get_menu_item_str (<menu_id>,<index>)
```

returns the string value of the menu's entry.

<menu_id>	the ID of the menu that you want to modify
<index>	the index of the menu entry

Remarks

The <index> is defined by the order in which the menu items are added within the init callback; it can't be changed afterwards.

Examples

```
on init
  declare ui_menu $menu
  add_menu_item ($menu, "First Entry",0)
  add_menu_item ($menu, "Second Entry",5)
  add_menu_item ($menu, "Third Entry",10)
  declare ui_button $button
end on

on ui_control ($button)
  message (get_menu_item_str (get_ui_id($menu),1))
end on
```

displays the message "Second Entry" when clicking on the button

See Also

$CONTROL_PAR_SELECTED_ITEM_IDX
$CONTROL_PAR_NUM_ITEMS
add_menu_item()
get_menu_item_value()
get_menu_item_visibility()
set_menu_item_str()
set_menu_item_value()
set_menu_item_visibility()

get_menu_item_value()

```
get_menu_item_value (<menu_id>,<index>)
```

returns the value of the menu's entry.

<menu_id>	the ID of the menu that you want to modify
<index>	the index of the menu entry

Remarks

The <index> is defined by the order in which the menu items are added within the init callback; it can't be changed afterwards.

Examples

```
on init
  declare ui_menu $menu
  add_menu_item ($menu, "First Entry",0)
  add_menu_item ($menu, "Second Entry",5)
  add_menu_item ($menu, "Third Entry",10)
  declare ui_button $button
end on

on ui_control ($button)
  message (get_menu_item_value (get_ui_id($menu),1))
end on
```

displays the number 5

See Also

$CONTROL_PAR_SELECTED_ITEM_IDX
$CONTROL_PAR_NUM_ITEMS
add_menu_item()
get_menu_item_str()
get_menu_item_visibility()
set_menu_item_str()
set_menu_item_value()
set_menu_item_visibility()

get_menu_item_visibility()

```
get_menu_item_visibility (<menu_id>,<index>)
```

returns 1 if the menu entry is visible, otherwise 0.

<menu_id>	the ID of the menu that you want to modify
<index>	the index of the menu entry

Remarks

The <index> is defined by the order in which the menu items are added within the init callback; it can't be changed afterwards.

Examples

```
on init
  declare ui_menu $menu
  add_menu_item ($menu, "First Entry",0)
  add_menu_item ($menu, "Second Entry",5)
  add_menu_item ($menu, "Third Entry",10)
  declare ui_button $button
end on

on ui_control ($button)
  message (get_menu_item_visibility (get_ui_id($menu),1))
end on
```

displays the value 1

See Also

$CONTROL_PAR_SELECTED_ITEM_IDX
$CONTROL_PAR_NUM_ITEMS
add_menu_item()
get_menu_item_str()
get_menu_item_value()
set_menu_item_str()
set_menu_item_value()
set_menu_item_visibility()

get_ui_id()

```
get_ui_id(<variable>)
```

retrieve the ID number of an ui control

Examples

```
on init
  declare ui_knob $Knob_1 (0,100,1)
  declare ui_knob $Knob_2 (0,100,1)
  declare ui_knob $Knob_3 (0,100,1)
  declare ui_knob $Knob_4 (0,100,1)

  declare ui_value_edit $Set(0,100,1)
  declare $a
  declare %knob_id[4]
  %knob_id[0] := get_ui_id ($Knob_1)
  %knob_id[1] := get_ui_id ($Knob_2)
  %knob_id[2] := get_ui_id ($Knob_3)
  %knob_id[3] := get_ui_id ($Knob_4)

end on

on ui_control ($Set)
  $a := 0
  while ($a < 4)
        set_control_par(%knob_id[$a],$CONTROL_PAR_VALUE,$Set)
        inc($a)
  end while
end on
```

store IDs in an array

See Also

set_control_par()

get_ui_wf_property()

```
get_ui_wf_property (<variable>,<property>,<index>)
```

returns the value of the waveform's different properties.

<variable>	the variable of the ui waveform
<property>	the following properties are available: $UI_WF_PROP_PLAY_CURSOR $UI_WF_PROP_FLAGS $UI_WF_PROP_TABLE_VAL $UI_WF_PROP_TABLE_IDX_HIGHLIGHT $UI_WF_PROP_MIDI_DRAG_START_NOTE
<index>	the index of the slice

Examples

```
on init
  declare $play_pos
  declare ui_waveform $Waveform(6,6)
  attach_zone ($Waveform,find_zone ("Test"),0)
end on

on note
  while ($NOTE_HELD = 1)
        $play_pos := get_event_par($EVENT_ID,$EVENT_PAR_PLAY_POS)
        set_ui_wf_property($Waveform,$UI_WF_PROP_PLAY_CURSOR,...
        0,$play_pos)
        message(get_ui_wf_property($Waveform,...
        $UI_WF_PROP_PLAY_CURSOR,0))
        wait (10000)
  end while
end on
```

displays the current play position value

See Also

set_ui_wf_property()
ui_waveform()
attach_zone()
find_zone()
Waveform Flag Constants
Waveform Property Constants

make_perfview

make_perfview
activates the performance view for the respective script

Remarks

`make_perfview` is only available in the init callback.

Examples

```
on init
  make_perfview
  set_script_title("Performance View")
  set_ui_height(6)
  message("")
end on
```

many performance view scripts start like this

See Also

`set_skin_offset()`
`set_ui_height()`

move_control()

move_control(<variable>,<x-position>,<y-position>)	
position ui elements in the standard Kontakt grid	
<variable>	the name of the ui control
<x-position>	the horizontal position of the control (0 to 6) in grids
<y-position>	the vertical position of the control (0 to 16) in grids

Remarks

- move_control() can be used in the init and other callbacks.
- Note that the usage of move_control() in other callbacks than the init callback is more cpu intensive, so handle with care,
- move_control(<variable>,0,0) will hide the ui element.

Examples

```
on init
  set_ui_height(3)
  declare ui_label $label (1,1)
  set_text ($label,"Move the wheel!")
  move_control ($label,3,6)
end on
on controller
  if ($CC_NUM = 1)
        move_control ($label,3,(%CC[1] * (-5) / (127)) + 6 )
  end if
end on
```

move a ui element with the modwheel (why you'd want to do that is up to you)

See Also

move_control_px()

move_control_px()

```
move_control_px(<variable>,<x-position>,<y-position>)
```

position ui elements in pixels

<variable>	the name of the ui control
<x-position>	the horizontal position of the control in pixels
<y-position>	the vertical position of the control in pixels

Remarks

- Once you position a control in pixel, you have to make all other adjustments in pixels too, i.e. you cannot change between "pixel" and "grid" mode for a specific control.
- `move_control_px()` can be used in the init and other callbacks.
- Note that the usage of `move_control_px()` in other callbacks than the init callback is more cpu intensive, so handle with care.
- move_control_px(<variable>,66,2) equals move_control(variable>,1,1)

Examples

```
on init
  declare ui_label $label (1,1)
  set_text ($label,"Move the wheel!")
  move_control_px ($label,66,2)
end on
on controller
  if ($CC_NUM = 1)
        move_control_px ($label,%CC[1]+66,2)
  end if
end on
```

transform cc values into pixel – might be useful for reference

See Also

```
move_control()
$CONTROL_PAR_POS_X
$CONTROL_PAR_POS_Y
```

set_control_help()

set_control_help(<variable>,<text>)	
assigns a text string to be displayed when hovering the ui control. The text will appear in Kontakt's info pane.	
<variable>	the name of the ui control
<text>	the info text to be displayed

Examples

```
on init
  declare ui_knob $Knob(0,100,1)
  set_control_help($Knob,"I'm the only knob, folks")
end on
```

set_control_help() in action

See Also

set_script_title()
$CONTROL_PAR_HELP

set_control_par()

set_control_par(<ui-ID>,<control-parameter>,<value>)	
change various parameters of the specified gui control	
<ui-ID>	the ID number of the ui control. You can retrieve the ID number with get_ui_id()
<control-parameter>	the control parameter variable like $CONTROL_PAR_WIDTH
<value>	the (integer) value

Remarks

set_control_par() comes in two additional flavors, set_control_par_str() for the usage with text strings and set_control_arr() for working with arrays.

Examples

```
on init
  declare ui_value_edit $test (0,100,$VALUE_EDIT_MODE_NOTE_NAMES)
  set_text ($test,"")
  set_control_par (get_ui_id($test),$CONTROL_PAR_WIDTH,45)
  move_control_px($test,100,10)
end on
```

changing the width of a value edit to 45 pixels. Note that you have to specify its position in pixels, too, once you enter "pixel-mode".

```
on init
  declare ui_label $test (1,1)
  set_text($test,"Text")
  set_control_par(get_ui_id($test),$CONTROL_PAR_TEXT_ALIGNMENT,1)
end on
```

center text in lables

See Also

get_control_par()
get_ui_id()

set_key_color()

```
set_key_color(<note-number>,<key-color-constant>)
```

sets the color of the specified key (i.e. MIDI note) on the Kontakt keyboard.

The following colors are available:

```
$KEY_COLOR_NONE {default value}
$KEY_COLOR_WHITE
$KEY_COLOR_YELLOW
$KEY_COLOR_GREEN
$KEY_COLOR_RED
$KEY_COLOR_CYAN
$KEY_COLOR_BLUE
$KEY_COLOR_BLACK {inverted}
```

Examples

```
on init
  declare $count
  while ($count < 128)
        set_key_color($count,$KEY_COLOR_WHITE)
        inc($count)
  end while
end on

on note
  select ($EVENT_VELOCITY)
        case 1 to 25
              set_key_color($EVENT_NOTE,$KEY_COLOR_BLUE)
        case 25 to 50
              set_key_color($EVENT_NOTE,$KEY_COLOR_CYAN)
        case 51 to 75
              set_key_color($EVENT_NOTE,$KEY_COLOR_GREEN)
        case 75 to 100
              set_key_color($EVENT_NOTE,$KEY_COLOR_YELLOW)
        case 100 to 127
              set_key_color($EVENT_NOTE,$KEY_COLOR_RED)
  end select
end on

on release
  set_key_color($EVENT_NOTE,$KEY_COLOR_WHITE)
end on
```

your own, private light organ

See Also

`set_control_help()`

set_knob_defval()

```
set_knob_defval(<variable>,<value>)
```

assign a default value to a knob to which the knob is reset when Cmd-clicking the knob.

Remarks

In order to assign a default value to a slider, use
`set_control_par(<ui-ID>,$CONTROL_PAR_DEFAULT_VALUE,<value>)`

Examples

```
on init
  declare ui_knob $Knob(-100,100,0)
  set_knob_defval ($Knob,0)
  $Knob := 0

  declare ui_slider $Slider (-100,100)
  set_control_par(get_ui_id($Slider),$CONTROL_PAR_DEFAULT_VALUE,0)
  $Slider := 0
end on
```

assigning default values to a knob and slider

See Also

`$CONTROL_PAR_DEFAULT_VALUE`

set_knob_label()

```
set_knob_label(<variable>,<text>)
```

assign a text string to a knob

Examples

```
on init
  declare !rate_names[18]
  !rate_names[0] := "1/128"
  !rate_names[1] := "1/64"
  !rate_names[2] := "1/32"
  !rate_names[3] := "1/16 T"
  !rate_names[4] := "3/64"
  !rate_names[5] := "1/16"
  !rate_names[6] := "1/8 T"
  !rate_names[7] := "3/32"
  !rate_names[8] := "1/8"
  !rate_names[9] := "1/4 T"
  !rate_names[10] := "3/16"
  !rate_names[11] := "1/4"
  !rate_names[12] := "1/2 T"
  !rate_names[13] := "3/8"
  !rate_names[14] := "1/2"
  !rate_names[15] := "3/4"
  !rate_names[16] := "4/4"
  !rate_names[17] := "Bar"

  declare ui_knob $Rate (0,17,1)
  set_knob_label($Rate,!rate_names[$Rate])

  read_persistent_var($Rate)
  set_knob_label($Rate,!rate_names[$Rate])
end on

on ui_control ($Rate)
  set_knob_label($Rate,!rate_names[$Rate])
end on
```

useful for displaying rhythmical values

See Also

$CONTROL_PAR_LABEL

set_knob_unit()

```
set_knob_unit(<variable>,<knob-unit-constant>)
```

assign a unit mark to a knob.

The following constants are available:

```
$KNOB_UNIT_NONE
$KNOB_UNIT_DB
$KNOB_UNIT_HZ
$KNOB_UNIT_PERCENT
$KNOB_UNIT_MS
$KNOB_UNIT_OCT
$KNOB_UNIT_ST
```

Examples

```
on init
  declare ui_knob $Time (0,1000,10)
  set_knob_unit ($Time,$KNOB_UNIT_MS)

  declare ui_knob $Octave (1,6,1)
  set_knob_unit ($Octave,$KNOB_UNIT_OCT)

  declare ui_knob $Volume (-600,600,100)
  set_knob_unit ($Volume,$KNOB_UNIT_DB)

  declare ui_knob $Scale (0,100,1)
  set_knob_unit ($Scale,$KNOB_UNIT_PERCENT)

  declare ui_knob $Tune (4300,4500,10)
  set_knob_unit ($Tune,$KNOB_UNIT_HZ)
end on
```

various knob unit marks

See Also

$CONTROL_PAR_UNIT

set_menu_item_str()

```
set_menu_item_str (<menu_id>,<index>,<string>)
```

sets the value of a menu entry.

<menu_id>	the ID of the menu that you want to modify
<index>	the menu entry's index
<string>	the menu entry's text

Remarks

The <index> is defined by the order in which the menu items are added within the init callback; it can't be changed afterwards.

Examples

```
on init
  declare ui_menu $menu
  declare ui_button $button
  add_menu_item ($menu, "First Entry",0)
  add_menu_item ($menu, "Second Entry",5)
  add_menu_item ($menu, "Third Entry",10)
end on

on ui_control ($button)
  set_menu_item_str (get_ui_id($menu),1,"Renamed")
end on
```

renaming the second menu entry

See Also

$CONTROL_PAR_SELECTED_ITEM_IDX
$CONTROL_PAR_NUM_ITEMS
add_menu_item()
get_menu_item_str()
get_menu_item_value()
get_menu_item_visibility()
set_menu_item_value()
set_menu_item_visibility()

set_menu_item_value()

```
set_menu_item_value (<menu_id>,<index>,<value>)
```

sets the value of a menu entry.

<menu_id>	the ID of the menu that you want to modify
<index>	the menu entry's index
<value>	the menu entry's value

Remarks

The <index> is defined by the order in which the menu items are added within the init callback; it can't be changed afterwards. The <value> is set by the third parameter of the add_menu_item() command.

Examples

```
on init
  declare ui_menu $menu
  add_menu_item ($menu, "First Entry",0)
  add_menu_item ($menu, "Second Entry",5)
  add_menu_item ($menu, "Third Entry",10)
  set_menu_item_value (get_ui_id($menu),1,20)
end on
```

changing the value of the second menu entry to 20

See Also

$CONTROL_PAR_SELECTED_ITEM_IDX
$CONTROL_PAR_NUM_ITEMS
add_menu_item()
get_menu_item_str()
get_menu_item_value()
get_menu_item_visibility()
set_menu_item_str()
set_menu_item_visibility()

set_menu_item_visibility()

```
set_menu_item_visibility (<menu_id>,<index>,<0,1>)
```

sets the value of a menu entry.

<menu_id>	the ID of the menu that you want to modify
<index>	the menu entry's index
<0,1>	invisible (**0**) / visible (**1**)

Remarks

The <index> is defined by the order in which the menu items are added within the init callback; it can't be changed afterwards. The <value> is set by the third parameter of the add_menu_item() command.
Add as many menu entries as you would possibly need within the init callback and then show or hide them dynamically by using set_menu_item_visibility().

Examples

```
on init
  declare ui_menu $menu
  declare ui_button $button
  add_menu_item ($menu, "First Entry",0)
  add_menu_item ($menu, "Second Entry",5)
  add_menu_item ($menu, "Third Entry",10)
end on

on ui_control ($button)
  set_menu_item_visibility (get_ui_id($menu),1,0)
end on
```

hiding the second menu enty

See Also

$CONTROL_PAR_SELECTED_ITEM_IDX
$CONTROL_PAR_NUM_ITEMS
add_menu_item()
get_menu_item_str()
get_menu_item_value()
get_menu_item_visibility()
set_menu_item_str()
set_menu_item_visibility()

set_table_steps_shown()

```
set_table_steps_shown(<variable>,<num-of-steps>)
```

changes the number of displayed columns in an ui table

<variable>	the name of the ui table
<num-of-steps>	the number of displayed steps

Examples

```
on init
  declare ui_table %table[32] (2,2,127)

  declare ui_value_edit $Steps (8,32,1)
  $Steps := 16
  set_table_steps_shown(%table,$Steps)

end on

on ui_control($Steps)
  set_table_steps_shown(%table,$Steps)
end on
```

changing the number of shown steps

See Also

ui_table

set_script_title()

```
set_script_title(<text>)
```

set the title of the script

Examples

```
on init
  make_perfview
  set_script_title("Performance View")
  set_ui_height(6)
  message("")
end on
```

many performance view scripts start like this

See Also

`make_perfview`

set_skin_offset()

```
set_skin_offset(<offset-in-pixel>)
```

offsets the chosen background tga file by the specified number of pixels

Remarks

If a background tga graphic file has been selected in the instrument options and this file is larger than the maximum height of the performance view, you can use this command to offset the background graphic, thus creating separate backgrounds for each of the script slots.

Examples

```
on init
  make_perfview
  set_ui_height(1)
end on

on controller
  if ($CC_NUM = 1)
        set_skin_offset(%CC[1])
  end if
end on
```

try this with the wallpaper called "Sunrise.tga" (Kontakt 4/presets/wallpaper/Sunrise.tga)

See Also

```
make_perfview
set_ui_height_px()
```

set_text()

```
set_text(<variable>,<text>)
```

when applied to a label: delete the text currently visible in the specified label and add new text
when applied to knobs, buttons, switches and value edits: set the name of the ui element

Examples

```
on init
  declare ui_label $label_1 (1,1)
  set_text ($label_1,"Small Label")

  declare ui_label $label_2 (3,6)
  set_text ($label_2,"Big Label")
  add_text_line ($label_2,"…with a second text line")
end on
```

two labels with different size

```
on init
  declare ui_label $label_1 (1,1)
  set_text ($label_1,"Small Label")
  hide_part ($label_1,$HIDE_PART_BG)
end on
```

hide the background of a label (also possible with other ui elements)

See Also

add_text_line()

set_ui_height()

set_ui_height(<height>)	
set the height of a script in grids	
<height>	the height of script in grids (1 to 8)

Remarks

Only possible in the init callback.

Examples

```
on init
  make_perfview
  set_script_title("Performance View")
  set_ui_height(6)
  message("")
end on
```

many performance view scripts start like this

See Also

set_ui_height_px()

set_ui_height_px()

set_ui_height_px(<height>)	
set the height of a script in pixel	
<height>	the height of script in pixel (50 to 540)

Remarks

Only possible in the init callback.

Examples

```
on init
  make_perfview
  declare const $SIZE := 1644 {size of tga file}
  declare const $NUM_SLIDES := 4 {amount of slides in tga file}

  declare ui_value_edit $Slide (1,$NUM_SLIDES,1)

  declare const $HEADER_SIZE := 93

  set_ui_height_px(($SIZE/$NUM_SLIDES)-$HEADER_SIZE)
  set_skin_offset (($Slide-1)*($SIZE/$NUM_SLIDES))

end on

on ui_control ($Slide)
  set_skin_offset (($Slide-1)*($SIZE/$NUM_SLIDES))
end on
```

try this with some of the wallpaper tga files of the Kontakt 4 Factory Library, e.g. /Kontakt 4 Library/Choir/Z - Samples/Wallpaper/pv_choir_bg.tga

See Also

set_ui_height()

set_ui_wf_property()

```
set_ui_wf_property (<variable>,<property>,<index>,<value>)
```

sets different properties for the waveform control

<variable>	the variable of the ui waveform
<property>	the following properties are available: $UI_WF_PROP_PLAY_CURSOR $UI_WF_PROP_FLAGS $UI_WF_PROP_TABLE_VAL $UI_WF_PROP_TABLE_IDX_HIGHLIGHT $UI_WF_PROP_MIDI_DRAG_START_NOTE
<index>	the index of the slice
<value>	the (integer) value

Examples

```
on init
  declare $play_pos
  declare ui_waveform $Waveform(6,6)
  attach_zone ($Waveform,find_zone("Test"),0)
end on

on note
  while ($NOTE_HELD = 1)
        $play_pos := get_event_par($EVENT_ID,$EVENT_PAR_PLAY_POS)
        set_ui_wf_property($Waveform,$UI_WF_PROP_PLAY_CURSOR,...
        0,$play_pos)
        wait (10000)
  end while
end on
```

attaches the zone "Test" to the waveform and displays a play cursor within the waveform as long as you play a note

See Also

get_ui_wf_property()
ui_waveform()
attach_zone()
find_zone()
Waveform Flag Constants
Waveform Property Constants

ui_button

```
declare ui_button $<variable-name>
```

create a user interface button

Remarks

`ui_button` is similiar to `ui_switch`, however it cannot be automated.

Examples

```
on init
  declare ui_button $free_sync_button
  $free_sync_button := 1
  set_text ($free_sync_button,"Sync")
  make_persistent ($free_sync_button)

  read_persistent_var($free_sync_button)
  if ($free_sync_button = 0)
        set_text ($free_sync_button,"Free")
  else
        set_text ($free_sync_button,"Sync")
  end if
end on

on ui_control ($free_sync_button)
  if ($free_sync_button = 0)
        set_text ($free_sync_button,"Free")
  else
        set_text ($free_sync_button,"Sync")
  end if
end on
```

a simple free/sync button implementation

See Also

`ui_switch`

ui_file_selector

```
declare ui_file_selector $<variable-name>
```

create a file selector

Remarks

There can be only one file selector per script slot.

Examples

```
on init
  declare $MIDISyncID
  declare ui_file_selector $fsMIDISelector
  set_control_par_str(get_ui_id($fsMIDISelector),...
  $CONTROL_PAR_BASEPATH,"/C:/NI Testpath")
  set_control_par_str(get_ui_id($fsMIDISelector),...
  $CONTROL_PAR_FILEPATH,"/C:/NI Testpath")
  set_control_par(get_ui_id($fsMIDISelector),...
  $CONTROL_PAR_FILE_TYPE,$NI_FILE_TYPE_MIDI)
  set_control_par(get_ui_id($fsMIDISelector),...
  $CONTROL_PAR_COLUMN_WIDTH,100)
  set_control_par(get_ui_id($fsMIDISelector),...
  $CONTROL_PAR_HEIGHT,100)
  set_control_par(get_ui_id($fsMIDISelector),...
  $CONTROL_PAR_WIDTH,550)
end on

on async_complete
  if ($NI_ASYNC_ID = $MIDISyncID)
        $MIDISyncID := -1
  end if
end on

on ui_control ($fsMIDISelector)
  $MIDISyncID := load_midi_file...
  (fs_get_filename(get_ui_id($fsMIDISelector),2))
  while ($MIDISyncID # -1)
        wait (100)
  end while
  message ("MIDI File loaded")
end on
```

loading MIDI files via ui file selector

See Also

$CONTROL_PAR_BASEPATH
$CONTROL_PAR_COLUMN_WIDTH

$CONTROL_PAR_FILEPATH
$CONTROL_PAR_FILE_TYPE
fs_get_filename()
fs_navigate()

ui_knob

```
declare ui_knob $<variable-name>(<min>,<max>,<display-ratio>)
```

create a user interface knob

<min>	the minimum value of the knob
<max>	the maximum value of the knob
<display-ratio>	the knob value is diveded by <display-ratio> for display purposes

Examples

```
on init
   declare ui_knob $Knob_1 (0,1000,1)
   declare ui_knob $Knob_2 (0,1000,10)
   declare ui_knob $Knob_3 (0,1000,100)
   declare ui_knob $Knob_4 (0,1000,20)
   declare ui_knob $Knob_5 (0,1000,-10)
end on
```

various display ratios

```
on init
   declare $count
   declare !note class[12]
   !note_class[0] := "C"
   !note_class[1] := "Db"
   !note class[2] := "D"
   !note class[3] := "Eb"
   !note class[4] := "E"
   !note class[5] := "F"
   !note_class[6] := "Gb"
   !note_class[7] := "G"
   !note class[8] := "Ab"
   !note class[9] := "A"
   !note class[10] := "Bb"
   !note class[11] := "B"
   declare !note_names [128]
   while ($count < 128)
          !note names[$count] := !note class[$count mod 12] & (($count/12)-2)
          inc ($count)
   end while

   declare ui_knob $Note (0,127,1)
   set knob label ($Note,!note names[$Note])
   make persistent ($Note)
    read persistent var($Note)
   set knob label ($Note,!note names[$Note])
end on
on ui_control ($Note)
   set knob label ($Note,!note names[$Note])
end on
```

knob displaying MIDI note names

See Also

set_knob_defval()
set_knob_label()
set_knob_unit()
ui_slider

ui_label

```
declare ui_label $<variable-name> (<width>,<height>)
```

create a user interface text label

<width>	the width of the label in grids
<height>	the height of the label in grids

Examples

```
on init
  declare ui_label $label_1 (1,1)
  set_text ($label_1,"Small Label")

  declare ui_label $label_2 (3,6)
  set_text ($label_2,"Big Label")
  add_text_line ($label_2,"…with a second text line")
end on
```

two labels with different size

```
on init
  declare ui_label $label_1 (1,1)
  set_text ($label_1,"Small Label")
  hide_part ($label_1,$HIDE_PART_BG)
end on
```

hide the background of a label (also possible with other ui elements)

See Also

set_text()
add_text_line()
hide_part()

ui_level_meter

```
declare ui_level_meter $<variable-name>
```

create a level meter

Remarks

With the level meter the scripter can create a volume display that allows him to reflect different gain stages of the instrument.

Examples

```
on init
  declare ui_level_meter $Level1
  declare ui_level_meter $Level2
  attach_level_meter (get_ui_id($Level1),-1,-1,0,-1)
  attach_level_meter (get_ui_id($Level2),-1,-1,1,-1)
end on
```

creating two volume meters, each one displaying one side of Kontakt's instrument output

See Also

$CONTROL_PAR_BG_COLOR
$CONTROL_PAR_OFF_COLOR
$CONTROL_PAR_ON_COLOR
$CONTROL_PAR_OVERLOAD_COLOR
$CONTROL_PAR_PEAK_COLOR
$CONTROL_PAR_VERTICAL
attach_level_meter()

ui_menu

```
declare ui_menu $<variable-name>
```

create a user interface menu

Examples

```
on init
  declare ui_menu $menu
  add_menu_item ($menu, "First Entry",0)
  add_menu_item ($menu, "Second Entry",1)
  add_menu_item ($menu, "Third Entry",2)
end on
```

a simple menu

```
on init
  declare $count
  declare ui_menu $menu

  $count := 1
  while ($count < 17)
        add_menu_item ($menu, "Entry Nr: " & $count,$count)
        inc ($count)
  end while
end on
```

create a menu with many entries in a jiffy

See Also

add_menu_item()

ui_slider

```
declare ui_slider $<variable-name> (<min>,<max>)
```

create a user interface slider

<min>	the minimum value of the slider
<max>	the maximum value of the slider

Examples

```
on init
  declare ui_slider $test (0,100)
  set_control_par(get_ui_id($test),$CONTROL_PAR_DEFAULT_VALUE,50)
end on
```

slider with default value

```
on init
  declare ui_slider $test (-100,100)
  $test := 0
  declare $id
  $id := get_ui_id($test)

  set_control_par($id,$CONTROL_PAR_MOUSE_BEHAVIOUR,2000)
  set_control_par($id,$CONTROL_PAR_DEFAULT_VALUE,0)
  set_control_par_str($id,$CONTROL_PAR_PICTURE,"K4_SLIDER_BIP_1")
end on
```

creating a bipolar slider by loading a different picture background

See Also

ui_knob

ui_switch

```
declare ui_switch $<variable-name>
```

create a user interface switch

Remarks

`ui_switch` is similiar to `ui_button`, however it's possible to automate the control.

Examples

```
on init
  declare ui_switch $rec_button
  set_text ($rec_button,"Record")
  declare $rec_button_id
  $rec_button_id:= get_ui_id ($rec_button)

  set_control_par ($rec_button_id,$CONTROL_PAR_WIDTH,60)
  set_control_par ($rec_button_id,$CONTROL_PAR_HEIGHT,20)

  set_control_par ($rec_button_id,$CONTROL_PAR_TEXT_ALIGNMENT,1)

  set_control_par ($rec_button_id,$CONTROL_PAR_POS_X,250)
  set_control_par ($rec_button_id,$CONTROL_PAR_POS_Y,5)
end on
```

switch with various settings utilizing set_control_par()

See Also

`ui_button`

ui_table

```
declare ui_table %<array>[columns](<width>,<height>,<range>)
```

create a user interface table

<width>	the width of the table in grids
<height>	the height of the table in grids
<range>	the range of the table. If negative values are used, a bipolar table is created

Examples

```
on init
  declare ui_table %table_uni[10] (2,2,100)
  declare ui_table %table_bi[10] (2,2,-100)
end on
```

unipolar and bipolar tables

```
on init
  declare ui_table %table[128] (5,2,100)
  declare ui_value_edit $Steps (1,127,1)
  $Steps := 16
  set_table_steps_shown (%table,$Steps)
end on
on ui_control ($Steps)
  set_table_steps_shown (%table,$Steps)
end on
```

changes the amount of shown steps (columns) in a table

See Also

set_table_steps_shown()

ui_text_edit

```
declare ui_text_edit @<variable-name>
```

create a text edit field

Remarks

This UI element allows the scripter to receive string variables from the user. The text edit's callback is triggered each time the user hits enter after typing or click anywhere else on the GUI.
Most GUI related control parameters (size, fonts etc.) are applicable to this element as well.

Examples

```
on init
  declare ui_text_edit @edit
end on

on ui_control (@edit)
  message (@edit)
end on
```

displaying user text in Kontakt's message field

See Also

`@ (string variable)`

ui_value_edit

```
declare ui_value_edit $<variable>(<min>,<max>,<$display-ratio>)
```

create a user interface number box

<min>	the minimum value of the value edit
<max>	the maximum value of the value edit
<display-ratio>	the value is diveded by <display-ratio> for display purposes You can also use $VALUE_EDIT_MODE_NOTE_NAMES to display note names

Examples

```
on init
  declare ui_value_edit $test (0,100,$VALUE_EDIT_MODE_NOTE_NAMES)
  set_text ($test,"")
  set_control_par (get_ui_id($test),$CONTROL_PAR_WIDTH,45)
  move_control_px($test,66,2)
end on

on note
  $test := $EVENT_NOTE
end on
```

value edit displaying note names

```
on init
  declare ui_value_edit $test (0,10000,1000)
  set_text ($test,"Value")
end on
```

value edit with three decimal spaces

See Also

$VALUE_EDIT_MODE_NOTE_NAMES
$CONTROL_PAR_SHOW_ARROWS

ui_waveform

declare ui_waveform $<variable>(<width>,<height>)	
create a waveform control to display zones and slices. Can also be used to control specific parameters per slice and for MIDI drag & drop functionality.	
<width>	the width of the waveform in grids
<height>	the height of the waveform in grids

Examples

```
on init
  declare ui_waveform $Waveform(6,6)
  attach_zone ($Waveform,find_zone(”Test”),0)
end on
```

displays the zone “Test” within the waveform control

See Also

set_ui_wf_property()
get_ui_wf_property()
attach_zone()
find_zone()
Waveform Flag Constants
Waveform Property Constants

Commands

abs()

```
abs(<expression>)
```

return the absolute value of an expression

Examples

```
on init
  declare $new_note
end on
on note
  $new_note := abs($EVENT_NOTE-127)
  change_note ($EVENT_ID,$new_note)
end on
```

a simple note inverter

See Also

inc()
dec()

by_marks()

by_marks(<bit-mark>)
a user defined group of events (or event IDs)

Remarks

by_marks() is a user defined group of events which can be set with set_event_mark(). It can be used with all commands which utilize event IDs like note_off(), change_tune() etc.

Examples

```
on note
  if ($EVENT_NOTE mod 12 = 0) {if played note is a c}
        set_event_mark($EVENT_ID,$MARK_1)
        change_tune(by_marks($MARK_1),%CC[1]*1000,0)
  end if
end on

on controller
  if($CC_NUM = 1)
        change_tune(by_marks($MARK_1),%CC[1]*1000,0)
  end if
end on
```

moving the mod wheel changes the tuning of all c's (C-2, C-1...C8)

See Also

set_event_mark()
$EVENT_ID
$ALL_EVENTS
$MARK_1 … $MARK_28

change_listener_par()

change_listener_par(<signal_type>,<parameter>)	
This command is similar to set_listener. It is used to change the parameters of the on listener callback. You can use this command in every callback.	
<signal_type>	the event which parameter should be changed: $NI_SIGNAL_TRANSP_STOP $NI_SIGNAL_TRANSP_START $NI_SIGNAL_TIMER_MS $NI_SIGNAL_TIMER_BEAT
<parameter>	the callback reacts (**1**) or doesn't react (**0**) to the corresponding signal type. When used in combination with $NI_SIGNAL_TIMER_MS it defines that the on listener callback is triggered **every** <parameter> microseconds. When used in combination with $NI_SIGNAL_TIMER_BEAT it defines how many times **per beat** the callback is triggered.

Examples

```
on init
  declare $mscount
  declare $beatcount

  set_listener ($NI_SIGNAL_TRANSP_STOP,1)
  set_listener ($NI_SIGNAL_TRANSP_START,1)
  set_listener ($NI_SIGNAL_TIMER_MS,1000)
  set_listener ($NI_SIGNAL_TIMER_BEAT,1)

  change_listener_par ($NI_SIGNAL_TIMER_MS,1000000)
end on

on listener
  select ($NI_SIGNAL_TYPE)
        case $NI_SIGNAL_TRANSP_STOP
              message ("Playback was stopped.")
        case $NI_SIGNAL_TRANSP_START
              message ("Playback was started.")
        case $NI_SIGNAL_TIMER_MS
              inc ($mscount)
              message ($mscount & "sec. / " & $beatcount & " beats")
        case $NI_SIGNAL_TIMER_BEAT
              inc ($beatcount)
              message ($mscount & "sec. / " & $beatcount & " beats")
  end select
end on
```

a timer and a bpm counter

See Also

set_listener
$NI_SIGNAL_TYPE

change_note()

```
change_note(<ID-number>,<note-number>)
```

change the note number of a specific note event

Remarks

- `change_note()` is only allowed in the note callback and only works before the first `wait()` statement. If the voice is already running, only the value of the variable changes.
- once the note number of a particular note event is changed, it becomes the new `$EVENT_NOTE`
- it is not possible to adress events via event groups like `$ALL_EVENTS`

Examples

```
on init
  declare %black_keys[5] := (1,3,6,8,10)
end on

on note
  if (search(%black_keys,$EVENT_NOTE mod 12) # -1)
        change_note($EVENT_ID,$EVENT_NOTE-1)
  end if
end on
```

constrain all notes to white keys, i.e. C major

See Also

`$EVENT_NOTE`
`change_velo()`

change_pan()

change_pan(<ID-number>,<panorama>,<relative-bit>)	
change the pan position of a specific note event	
<ID-number>	the ID number of the note event to be changed
<panorama>	the pan position of the note event, from -1000 (left) to 1000 (right)
<relative-bit>	If the relative bit is set to **0**, the amount is **absolute**, i.e. the amount overwrites any previous set values of that event. If set to **1**, the amount is **relative** to the actual value of the event. The different implications are only relevant with more than one change_pan() statement applied to the same event.

Remarks

- change_pan() works on a note event level and does not change any panorama settings in the instrument itself. It is also not related to any MIDI modulations regarding panorama.

Examples

```
on init
  declare $pan_position
end on
on note
  $pan_position := ($EVENT_NOTE * 2000 / 127) - 1000
  change_pan ($EVENT_ID,$pan_position,0)
end on
```

panning the entire key range from left to right, i.e. C-2 all the way left, G8 all the way right

```
on note
  if ($EVENT_NOTE < 60)
        change_pan ($EVENT_ID,1000,0)
        wait(500000)
        change_pan ($EVENT_ID,-1000,0) {absolute, pan is at -1000}
  else
        change_pan ($EVENT_ID,1000,1)
        wait(500000)
        change_pan ($EVENT_ID,-1000,1) {relative, pan is at 0}
  end if
end on
```

notes below C3 utilize a relative-bit of 0, C3 and above utilize a relative bit of 1

See Also

change_vol()
change_tune()

change_tune()

change_tune(<ID-number>,<tune-amount>,<relative-bit>)	
change the tuning of a specific note event in millicent	
<ID-number>	the ID number of the note event to be changed
<tune-amount>	the tune amount in millicents, so 100000 equals 100 cent (i.e. a half tone)
<relative-bit>	If the relative bit is set to **0**, the amount is **absolute**, i.e. the amount overwrites any previous set values of that event. If it is set to **1**, the amount is **relative** to the actual value of the event. The different implications are only relevant with more than one change_tune() statement applied to the same event.

Remarks

- change_tune() works on a note event level and does not change any tune settings in the instrument itself. It is also not related to any MIDI modulations regarding tuning.

Examples

```
on init
  declare $tune_amount
end on

on note
  $tune_amount := random(-50000,50000)
  change_tune ($EVENT_ID,$tune_amount,1)
end on
```

randomly detune each note by ± 50 cent

See Also

change_vol()
change_pan()

change_velo()

```
change_velo(<ID-number>,<velocity>)
```

change the velocity of a specific note event

Remarks

- `change_velo()` is only allowed in the note callback and only works before the first `wait()` statement. If the voice is already running, only the value of the variable changes.
- once the velocity of a particular note event is changed, it becomes the new `$EVENT_VELOCITY`
- it is not possible to adress events via event groups like `$ALL_EVENTS`

Examples

```
on note
  change_velo ($EVENT_ID,100)
  message($EVENT_VELOCITY)
end on
```

all velocities are set to 100. Note that `$EVENT_VELOCITY` *will also change to 100.*

See Also

`$EVENT_VELOCITY`
`change_note()`

change_vol()

change_vol(<ID-number>,<volume>,<relative-bit>)	
change the volume of a specific note event in millidecibel	
<ID-number>	the ID number of the note event to be changed
<volume>	the volume change in millidecibel
<relative-bit>	If the relative bit is set to **0**, the amount is **absolute**, i.e. the amount overwrites any previous set values of that event. If it is set to **1**, the amount is **relative** to the actual value of the event. The different implications are only relevant with more than one change_vol() statement applied to the same event.

Remarks

- change_vol() works on a note event level and does not change any tune settings in the instrument itself. It is also not related to any MIDI modulations regarding volume (e.g. MIDI CC7).

Examples

```
on init
  declare $vol_amount
end on

on note
  $vol_amount := (($EVENT_VELOCITY - 1) * 12000/126) - 6000
  change_vol ($EVENT_ID,$vol_amount,1)
end on
```

a simple dynamic expander: lightly played notes will be softer, harder played notes will be louder

See Also

change_tune()
change_pan()
fade_in()
fade_out()

dec()

```
dec(<expression>)
```

decrement an expression by 1

Examples

```
on init
  declare ui_button $Reset
  declare $volume
end on

on ui_control ($Reset)
  $volume := 0
  $Reset := 0
end on

on note
  dec($volume)
  change_vol($EVENT_ID,$volume*1000,0)
end on
```

note fader: each played note is 1dB softer than the previous one

See Also

abs()
inc()

delete_event_mark()

delete_event_mark(<ID-number>,<bit-mark>	
delete an event mark, i.e. ungroup the specified event from an event group	
<ID-number> <bit-mark>	the ID number of the event to be ungrouped here you can enter one of 28 marks from $MARK_1 to $MARK_28 which is addressed to the event. You can also address more than one mark to a single event, either by typing the command or by using the operator +.

See Also

set_event_mark()
by_marks()
$EVENT_ID
$ALL_EVENTS
$MARK_1 … $MARK_28

event_status()

```
event_status(<ID-number>)
```

retrieve the status of a particular note event (or MIDI event in the multi script)

The note can either be active, then this function returns
`$EVENT_STATUS_NOTE_QUEUE` (or `$EVENT_STATUS_MIDI_QUEUE` in the multi script)

or inactive, then the function returns
`$EVENT_STATUS_INACTIVE`

Remarks

`event_status()` can be used to find out if a note event is still "alive" or not.

Examples

```
on init
  declare %key_id[128]
end on

on note
  if (event_status(%key_id[$EVENT_NOTE])= $EVENT_STATUS_NOTE_QUEUE)
        fade_out(%key_id[$EVENT_NOTE],10000,1)
  end if
  %key_id[$EVENT_NOTE] := $EVENT_ID
end on
```

limit the number of active note events to one per MIDI key

See Also

`$EVENT_STATUS_INACTIVE`
`$EVENT_STATUS_NOTE_QUEUE`
`$EVENT_STATUS_MIDI_QUEUE`
`get_event_ids()`

exit

exit
immediately stops a callback or exits a function

Remarks

- `exit` is a very "strong" command. Be very careful when using it, especially when dealing with larger scripts.
- If used with a function, `exit` only quits the function but not the entire callback.

Examples

```
on note
  if (not(in_range($EVENT_NOTE,60,71)))
        exit
  end if
  {from here on, only notes between C3 to B3 will be processed}
end on
```

useful for quickly setting up key ranges to be affected by the script

See Also

`wait()`

fade_in()

fade_in(<ID-number>,<fade-time>)	
perform a fade-in for a specific note event	
<ID-number>	the ID number of the note event to be faded in
<fade-time>	the fade-in time in microseconds

Examples

```
on init
  declare $note_1_id
  declare $note_2_id
end on

on note
  $note_1_id := play_note($EVENT_NOTE+12,$EVENT_VELOCITY,0,-1)
  $note_2_id := play_note($EVENT_NOTE+19,$EVENT_VELOCITY,0,-1)

  fade_in ($note_1_id,1000000)
  fade_in ($note_2_id,5000000)
end on
```

fading in the first two harmonics

See Also

change_vol()
fade_out()

fade_out()

fade_out(<ID-number>,<fade-time>,<stop-voice>)	
perform a fade-out for a specific note event	
<ID-number>	the ID number of the note event to be faded in
<fade-time>	the fade-in time in microseconds
<stop_voice>	If set to **1**, the voice is stopped after the fade out. If set to **0**, the voice will still be running after the fade out

Examples

```
on controller
  if ($CC_NUM = 1)
        if (%CC[1] mod 2 # 0)
              fade_out($ALL_EVENTS,5000,0)
        else
              fade_in($ALL_EVENTS,5000)
        end if
  end if
end on
```

use the mod whel on held notes to create a stutter effect

```
on controller
  if ($CC_NUM = 1)
        fade_out($ALL_EVENTS,5000,1)
  end if
end on
```

a custom "All Sound Off" implementation triggered by the mod wheel

See Also

change_vol()
fade_in()

get_event_ids()

get_event_ids(<array-name>)	
fills the specified array with all active event IDs. The command overwrites all existing values as long as there are events and writes 0 if no events are active anymore.	
<array-name>	array to be filled with active event IDs

Examples

```
on init
   declare const $ARRAY_SIZE := 500
   declare %test_array[$ARRAY_SIZE]
   declare $a
   declare $note_count
end on

on note
   get_event_ids(%test_array)
   $a := 0
   $note_count := 0
   while($a < $ARRAY_SIZE and %test_array[$a] # 0)
          inc($note_count)
          inc($a)
   end while
   message("Active Events: " & $note_count)
end on
```

monitoring the number of active events

See Also

event_status()
ignore_event()

get_event_par()

```
get_event_par(<ID-number>,<parameter>)
```

return the value of a specific event parameter of the specified event

<ID-number>	the ID number of the event
<parameter>	the event parameter, either one of four freely assignable event parameter: $EVENT_PAR_0 $EVENT_PAR_1 $EVENT_PAR_2 $EVENT_PAR_3 or the "built-in" parameters of a note event: $EVENT_PAR_VOLUME $EVENT_PAR_PAN $EVENT_PAR_TUNE $EVENT_PAR_NOTE $EVENT_PAR_VELOCITY $EVENT_PAR_SOURCE $EVENT_PAR_PLAY_POS

Remarks

A note event always "carries" certain information like the note number, the played velocity, but also Volume, Pan and Tune. With `set_event_par()`, you can set either these parameters or use the freely assignable parameters like `$EVENT_PAR_0`. This is especially useful when chaining scripts, i.e. set an event parameter for an event in slot 1, then retrieve this information in slot 2 by using `get_event_par()`.
The event parameters are not influenced by the system scripts anymore.

Examples

```
on note
  message(get_event_par($EVENT_ID,$EVENT_PAR_NOTE))
end on
```

the same functionality as message($EVENT_NOTE)

```
on note
  message(get event par($EVENT ID,$EVENT PAR SOURCE))
end on
```

check if the event comes from outside (-1) or if it was created in one of the five script slots (0-4)

See Also

```
set_event_par()
ignore_event()
set_event_par_arr()
get_event_par_arr()
```

get_event_par_arr()

get_event_par_arr(<ID-number>,<parameter>,<group-index>)	
special form of get_event_par(), used to retrieve the group allow state of the specified event	
<ID-number>	the ID number of the note event
<parameter>	in this case, only $EVENT_PAR_ALLOW_GROUP
<group-index>	the index of the group for changing the specified note's group allow state

Remarks

- get_event_par_arr() is a special form (or to be more precise, it's the array variant) of get_event_par(). It is used to retrieve the allow state of a specific event. If will return 1, if the specified group is allowed and 0 if it's disaalowed.

Examples

```
on note
  disallow_group($ALL_GROUPS)
  set_event_par_arr($EVENT_ID,$EVENT_PAR_ALLOW_GROUP,1,0)
end on
```

allowing only the first group, same as allow_group(0)

See Also

set_event_par_arr()
get_event_par()
$EVENT_PAR_ALLOW_GROUP

get_folder()

get_folder(<path-variable>)	
return the path specified with the built-in path variable	
<path-variable>	the following path variables are available: $GET_FOLDER_LIBRARY_DIR the library directory (as set in Options/Load-Import) $GET_FOLDER_FACTORY_DIR the factory folder of Kontakt, mainly used for loading factory IR samples Note: this is not the factory library folder! $GET_FOLDER_PATCH_DIR the directory in which the patch was saved. If the patch was not saved before, an empty string is returned.

Remarks

All paths for saving or loading anything as well as setting any kind of path for the ui_file_selector element should be formatted in the following way: **/C:/myFolder/mysubfolder**
It is necessary to use a "/" (slash character) and a "\" (backslash character) as a folder separator, even for Windows based systems. In addition to that, the full path has to start with a "/". All paths returned by the `get_folder()` and the `fs_get_filename()` commands are in this format.

Examples

```
on init
  message(get_folder($GET_FOLDER_LIBRARY_DIR))
end on
```

where did I put my factory library?

See Also

`load_ir_sample()`
`load_midi_file()`
`save_midi_file()`

ignore_controller

```
ignore_controller
```

ignore a controller event in a controller callback

Examples

```
on controller
  if ($CC_NUM = 1)
        ignore_controller
        set_controller($VCC_MONO_AT,%CC[1]
  end if
end on
```

transform the mod wheel into aftertouch

See Also

ignore_event()
set_controller()

ignore_event()

```
ignore_event(<ID-number>)
```

ignore a note event in a note on or note off callback

Remarks

- If you ignore an event, any volume, tune or pan information is lost. You can however retrieve this infomation with `get_event_par()`, see the two examples below.
- `ignore_event()` is a very "strong" command. Always check if you can get the same results with the various `change_xxx()` commands without having to ignore the event.

Examples

```
on note
  ignore_event($EVENT_ID)
  wait (500000)
  play_note($EVENT_NOTE,$EVENT_VELOCITY,0,-1)
end on
```

delaying all notes by 0.5s. Not bad, but if you for example insert a microtuner before this script, the tuning information will be lost

```
on init
  declare $new_id
end on

on note
  ignore_event($EVENT_ID)
  wait (500000)
  $new_id := play_note($EVENT_NOTE,$EVENT_VELOCITY,0,-1)

  change_vol($new_id,get_event_par($EVENT_ID,$EVENT_PAR_VOLUME),1)
  change_tune($new_id,get_event_par($EVENT_ID,$EVENT_PAR_TUNE),1)
  change_pan($new_id,get_event_par($EVENT_ID,$EVENT_PAR_PAN),1)
end on
```

better: the tuning (plus volume and pan to be precise) information is retrieved and applied to the played note

See Also

```
ignore_controller
get_event_par()
```

inc()

```
inc(<expression>)
```

increment an expression by 1

Examples

```
on init
  declare $count
  declare ui_table %table[100] (6,2,100)
  while ($count < num_elements(%table))
        %table[$count] := 50
        inc ($count)
  end while
end on
```

initializing a table with a specific value by using a while loop

See Also

abs()
dec()
while()

load_midi_file()

load_midi_file(<path>)	
loads a MIDI file	
<path>	the absolute path of the MIDI file.

Remarks

- There can be only one loaded MIDI file per NKI. It is accessible from each script slot.
- Loading a new MIDI file will overwrite all changes made to a previously loaded MIDI file without any warning. Please make sure to save any changes.
- As this command needs an absolute path and MIDI files are not stored within the Resource Container, you need to use a command like `get_folder ($GET_FOLDER_PATCH_DIR)` or `get_folder ($GET_FOLDER_LIBRARY_DIR)` to load your file correctly.
- This command returns a unique value after is has finished its action. Please use it in combination with `$NI_ASYNC_ID` within the `on_async_complete` callback to avoid synchronization issues.

Example

```
on init
  declare $sync_id
  declare ui_button $Load
end on

on async_complete
  if ($NI_ASYNC_ID = $sync_id)
        $sync_id := -1
  end if
end on

on ui_control ($Load)
  $sync_id := load_midi_file (get_folder ($GET_FOLDER_PATCH_DIR)...
  & "midifile.mid")
  while ($sync_id # -1)
        wait (1)
  end while
end on
```

Loading a MIDI file.

See Also

`$NI_ASYNC_ID`
`on async_complete`
`save_midi_file()`

lsb()

```
lsb(<value>)
```

return the LSB portion (least significant byte) of a 14 bit value

Examples

```
on rpn
  message(lsb($RPN_VALUE))
end on
```

commonly used when working with rpn and nrpn messages

```
on init
  declare ui_value_edit $Value (0,16383,1)
end on

on ui_control ($Value)
  message("MSB: " & msb($Value) & " - LSB: " & lsb($Value))
end on
```

Understanding MSB and LSB

See Also

msb()
$RPN_ADDRESS
$RPN_VALUE

make_persistent()

```
make_persistent(<variable>)
```

retain the value of a variable when a patch is saved

Remarks

- The state of the variable is saved not only with the patch (or multi or host chunk), but also when a script is saved as a Kontakt preset (.nkp file).
- The state of the variables is read at the end of the init callback. To load a stored value manually within the init callback, use `read_persistent_var()`.
- When replacing script code by copy and replacing the text, the values of persistent variables is also retained. Sometimes, when working on more complex scripts, you'll want to "flush" the variables by resetting the script, that is applying an empty script in the respective slot.

Examples

```
on init
  declare ui_knob $Preset (1,10,1)
  make_persistent ($Preset)
end on
```

user interface elements like knobs should usually retain their value when reloading the instrument

See Also

`read_persistent_var()`

message()

```
message(<variable/text>)
```

display text in the status line of Kontakt

Remarks

- The message command is intended to be used for debugging and testing while programming a script. Since there is only one status line in Kontakt, it should not be used as a generic means of communication with the user, use a label instead.
- Make it a habit to write `message("")` in the init callback. You can then be sure that all previous messages (by the script or by the system) are deleted and you see only new messages.

Examples

```
on init
  message("Hello, world!")
end on
```

the inevitable implementation of "Hello, world!" in KSP

```
on note
  message("Note " & $EVENT_NOTE & " received at " & ...
  $ENGINE_UPTIME  & " milliseconds")
end on
```

concatenating elements in a message() command

See Also

$ENGINE_UPTIME
$KSP_TIMER
reset_ksp_timer
declare ui_label
set_text()

mf_get_byte_one()

```
mf_get_byte_one()
```

returns the value of the first byte of the currently selected MIDI event

Remarks

- If the start or end of the file / track is reached, `mf_get_byte_one()`, `mf_get_byte_two()`, `mf_get_channel()`, `mf_get_command()`, `mf_get_pos()` and `mf_get_track_idx()` return 0. In that respect, checking whether `mf_get_command()` returns 0 would be the way to check whether the cursor is out of range.

Example

Please see the “Working with MIDI files” section for an explanation on how to handle MIDI files.

See Also

```
load_midi_file()
mf_get_byte_two()
mf_get_channel()
mf_get_command()
mf_get_pos()
mf_get_track_idx()
mf_set_byte_one()
mf_set_byte_two()
mf_set_channel()
mf_set_command()
mf_set_pos()
save_midi_file()
```

mf_get_byte_two()

```
mf_get_byte_two()
```

returns the value of the second byte of the currently selected MIDI event

Remarks

- If the start or end of the file / track is reached, `mf_get_byte_one()`, `mf_get_byte_two()`, `mf_get_channel()`, `mf_get_command()`, `mf_get_pos()` and `mf_get_track_idx()` return 0. In that respect, checking whether `mf_get_command()` returns 0 would be the way to check whether the cursor is out of range.

Example

Please see the “Working with MIDI files” section for an explanation on how to handle MIDI files.

See Also

```
load_midi_file()
mf_get_byte_one()
mf_get_channel()
mf_get_command()
mf_get_pos()
mf_get_track_idx()
mf_set_byte_one()
mf_set_byte_two()
mf_set_channel()
mf_set_command()
mf_set_pos()
save_midi_file()
```

mf_get_channel()

mf_get_channel()
returns the channel number of the currently selected MIDI event

Remarks

- Please note that KONTAKT internally uses values of 0 to 15 for its MIDI channels, not 1 to 16.
- If the start or end of the file / track is reached, `mf_get_byte_one()`, `mf_get_byte_two()`, `mf_get_channel()`, `mf_get_command()`, `mf_get_pos()` and `mf_get_track_idx()` return 0. In that respect, checking whether `mf_get_command()` returns 0 would be the way to check whether the cursor is out of range.

Example

Please see the “Working with MIDI files” section for an explanation on how to handle MIDI files.

See Also

```
load_midi_file()
mf_get_byte_one()
mf_get_byte_two()
mf_get_command()
mf_get_pos()
mf_get_track_idx()
mf_set_byte_one()
mf_set_byte_two()
mf_set_channel()
mf_set_command()
mf_set_pos()
save_midi_file()
```

mf_get_command()

mf_get_byte_one()
returns the command type of the currently selected MIDI event

Remarks

- You can compare the returned value to internal variables like `$MIDI_COMMAND_NOTE_ON` or `$MIDI_COMMAND_PITCH_BEND` to find out the current command type.
- If the start or end of the file / track is reached, `mf_get_byte_one()`, `mf_get_byte_two()`, `mf_get_channel()`, `mf_get_command()`, `mf_get_pos()` and `mf_get_track_idx()` return 0. In that respect, checking whether `mf_get_command()` returns 0 would be the way to check whether the cursor is out of range.

Example

Please see the “Working with MIDI files” section for an explanation on how to handle MIDI files.

See Also

```
load_midi_file()
mf_get_byte_one()
mf_get_byte_two()
mf_get_channel()
mf_get_pos()
mf_get_track_idx()
mf_set_byte_one()
mf_set_byte_two()
mf_set_channel()
mf_set_command()
mf_set_pos()
save_midi_file()
```

mf_get_pos()

mf_get_pos()
returns the position of the currently selected MIDI event in ticks

Remarks

- If the start or end of the file / track is reached, `mf_get_byte_one()`, `mf_get_byte_two()`, `mf_get_channel()`, `mf_get_command()`, `mf_get_pos()` and `mf_get_track_idx()` return 0. In that respect, checking whether `mf_get_command()` returns 0 would be the way to check whether the cursor is out of range.

Example

Please see the “Working with MIDI files” section for an explanation on how to handle MIDI files.

See Also

```
load_midi_file()
mf_get_byte_one()
mf_get_byte_two()
mf_get_channel()
mf_get_command()
mf_get_track_idx()
mf_set_byte_one()
mf_set_byte_two()
mf_set_channel()
mf_set_command()
mf_set_pos()
save_midi_file()
```

mf_get_track_idx()

```
mf_get_track_idx()
```

returns the track index of the currently selected MIDI event

Remarks

- If the start or end of the file / track is reached, `mf_get_byte_one()`, `mf_get_byte_two()`, `mf_get_channel()`, `mf_get_command()`, `mf_get_pos()` and `mf_get_track_idx()` return 0. In that respect, checking whether `mf_get_command()` returns 0 would be the way to check whether the cursor is out of range.

Example

Please see the “Working with MIDI files” section for an explanation on how to handle MIDI files.

See Also

```
load_midi_file()
mf_get_byte_one()
mf_get_byte_two()
mf_get_channel()
mf_get_command()
mf_get_pos()
mf_set_byte_one()
mf_set_byte_two()
mf_set_channel()
mf_set_command()
mf_set_pos()
save_midi_file()
```

mf_get_first()

mf_get_first(<track-index>)	
jumps to the first event in the MIDI track	
<track-index>	the number of the track you want to edit. **-1** refers to the whole file.

Remarks

- There is no built-in "player" functionality. You can only query parameters of certain events and of the MIDI file itself.
- You can only work on one MIDI event at a time.

Example

Please see the "Working with MIDI files" section for an explanation on how to handle MIDI files.

See Also

```
load_midi_file()
mf_get_next()
mf_get_next_at()
mf_get_num_tracks()
mf_get_last()
mf_get_prev()
mf_get_prev_at()
save_midi_file()
```

mf_get_next()

mf_get_next(<track-index>)	
jumps to the next event in the MIDI track	
<track-index>	the number of the track you want to edit. **-1** refers to the whole file.

Remarks

- There is no built-in "player" functionality. You can only query parameters of certain events and of the MIDI file itself.
- You can only work on one MIDI event at a time.

Example

Please see the "Working with MIDI files" section for an explanation on how to handle MIDI files.

See Also

```
load_midi_file()
mf_get_first()
mf_get_next_at()
mf_get_num_tracks()
mf_get_last()
mf_get_prev()
mf_get_prev_at()
save_midi_file()
```

mf_get_next_at()

mf_get_next_at(<track-index>,<pos>)	
jumps to the next event in the MIDI track right after this certain position.	
<track-index> <pos>	the number of the track you want to edit. **-1** refers to the whole file. position in ticks

Remarks

- There is no built-in "player" functionality. You can only query parameters of certain events and of the MIDI file itself.
- You can only work on one MIDI event at a time.

Example

Please see the "Working with MIDI files" section for an explanation on how to handle MIDI files.

See Also

```
load_midi_file()
mf_get_first()
mf_get_next()
mf_get_num_tracks()
mf_get_last()
mf_get_prev()
mf_get_prev_at()
save_midi_file()
```

mf_get_num_tracks()

```
mf_get_num_tracks()
```

returns the number of tracks in a MIDI file.

Example

Please see the “Working with MIDI files” section for an explanation on how to handle MIDI files.

See Also

```
load_midi_file()
mf_get_first()
mf_get_next()
mf_get_next_at()
mf_get_last()
mf_get_prev()
mf_get_prev_at()
save_midi_file()
```

mf_get_last()

mf_get_last(<track-index>)	
jumps to the last event in the MIDI track	
<track-index>	the number of the track you want to edit. **-1** refers to the whole file.

Remarks

- There is no built-in "player" functionality. You can only query parameters of certain events and of the MIDI file itself.
- You can only work on one MIDI event at a time.

Example

Please see the "Working with MIDI files" section for an explanation on how to handle MIDI files.

See Also

```
load_midi_file()
mf_get_first()
mf_get_next()
mf_get_next_at()
mf_get_num_tracks()
mf_get_prev()
mf_get_prev_at()
save_midi_file()
```

mf_get_prev()

mf_get_prev(<track-index>)	
jumps to the previous event in the MIDI track	
<track-index>	the number of the track you want to edit. **-1** refers to the whole file.

Remarks

- There is no built-in "player" functionality. You can only query parameters of certain events and of the MIDI file itself.
- You can only work on one MIDI event at a time.

Example

Please see the "Working with MIDI files" section for an explanation on how to handle MIDI files.

See Also

```
load_midi_file()
mf_get_first()
mf_get_next()
mf_get_next_at()
mf_get_num_tracks()
mf_get_last()
mf_get_prev_at()
save_midi_file()
```

mf_get_prev_at()

mf_get_prev_at(<track-index>,<pos>)	
jumps to the previous event in the MIDI track right before this certain position	
<track-index> <pos>	the number of the track you want to edit. **-1** refers to the whole file. position in ticks

Remarks

- There is no built-in "player" functionality. You can only query parameters of certain events and of the MIDI file itself.
- You can only work on one MIDI event at a time.

Example

Please see the "Working with MIDI files" section for an explanation on how to handle MIDI files.

See Also

```
load_midi_file()
mf_get_first()
mf_get_next()
mf_get_next_at()
mf_get_num_tracks()
mf_get_last()
mf_get_prev()
save_midi_file()
```

mf_set_byte_one()

mf_set_byte_one(<par>)	
sets the value of the first byte of the currently selected MIDI event	
<par>	the value of the first byte

Example

Please see the “Working with MIDI files” section for an explanation on how to handle MIDI files.

See Also

```
load_midi_file()
mf_get_type_one()
mf_get_byte_two()
mf_get_channel()
mf_get_command()
mf_get_pos()
mf_get_track_idx()
mf_set_byte_two()
mf_set_channel()
mf_set_command()
mf_set_pos()
save_midi_file()
```

mf_set_byte_two()

mf_set_byte_two(<par>)	
sets the value of the second byte of the currently selected MIDI event	
<par>	the value of the second byte

Example

Please see the “Working with MIDI files” section for an explanation on how to handle MIDI files.

See Also

```
load_midi_file()
mf_get_type_one()
mf_get_byte_two()
mf_get_channel()
mf_get_command()
mf_get_pos()
mf_get_track_idx()
mf_set_byte_one()
mf_set_channel()
mf_set_command()
mf_set_pos()
save_midi_file()
```

mf_set_channel()

mf_set_byte_one(<par>)	
changes the MIDI channel of the currently selected MIDI event	
<par>	the channel number (0 to 15)

Remarks

- Please note that KONTAKT internally uses values of 0 to 15 for its MIDI channels, not 1 to 16.

Example

Please see the “Working with MIDI files” section for an explanation on how to handle MIDI files.

See Also

```
load_midi_file()
mf_get_type_one()
mf_get_byte_two()
mf_get_channel()
mf_get_command()
mf_get_pos()
mf_get_track_idx()
mf_set_byte_one()
mf_set_byte_two()
mf_set_command()
mf_set_pos()
save_midi_file()
```

mf_set_command()

mf_set_command(<par>)	
changes the command type of the currently selected MIDI event	
<par>	the command type

Remarks

- You use internal variables like $MIDI_COMMAND_NOTE_ON or $MIDI_COMMAND_PITCH_BEND to make things easier.

Example

Please see the “Working with MIDI files” section for an explanation on how to handle MIDI files.

See Also

```
load_midi_file()
mf_get_type_one()
mf_get_byte_two()
mf_get_channel()
mf_get_command()
mf_get_pos()
mf_get_track_idx()
mf_set_byte_one()
mf_set_byte_two()
mf_set_channel()
mf_set_pos()
save_midi_file()
```

mf_set_pos()

mf_set_pos(<par>)	
changes the position of the currently selected MIDI event	
<par>	position in ticks

Example

Please see the “Working with MIDI files” section for an explanation on how to handle MIDI files.

See Also

```
load_midi_file()
mf_get_type_one()
mf_get_byte_two()
mf_get_channel()
mf_get_command()
mf_get_pos()
mf_get_track_idx()
mf_set_byte_one()
mf_set_byte_two()
mf_set_channel()
mf_set_command()
save_midi_file()
```

ms_to_ticks()

```
ms_to_ticks(<microseconds>)
```

converts a microseconds value into ticks

Examples

```
on init
  declare ui_label $bpm(1,1)
  set_text($bpm,ms_to_ticks(60000000)/960)
end on
```

displaying the current host tempo

See Also

ticks_to_ms()
$NI_SONG_POSITION

msb()

```
msb(<value>)
```

return the MSB portion (most significant byte) of a 14 bit value

Examples

```
on rpn
  message(msb($RPN_VALUE))
end on
```

commonly used when working with rpn and nrpn messages

```
on init
  declare ui_value_edit $Value (0,16383,1)
end on

on ui_control ($Value)
  message("MSB: " & msb($Value) & " - LSB: " & lsb($Value))
end on
```

Understanding MSB and LSB

See Also

lsb()
$RPN_ADDRESS
$RPN_VALUE

note_off()

```
note_off(<ID-number>)
```

send a note off message to a specific note

`<ID-number>`	the ID number of the note event

Remarks

- `note_off()` is equivalent to releasing a key, thus it will always trigger a release callback as well as the release portion of a volume envelope. Notice the difference between `note_off()` and `fade_out`, since `fade_out()` works more on a voice level

Examples

```
on controller
  if ($CC_NUM = 1)
        note_off($ALL_EVENTS)
  end if
end on
```

a custom "All Notes Off" implementation triggered by the mod wheel

```
on init
  declare polyphonic $new_id
end on

on note
  ignore_event($EVENT_ID)
  $new_id := play_note($EVENT_NOTE,$EVENT_VELOCITY,0,0)
end on

on release
  ignore_event($EVENT_ID)
  wait(200000)
  note_off($new_id)
end on
```

delaying the release of each note by 200ms

See Also

`fade_out()`
`play_note()`

output_channel_name()

output_channel_name(<output-number>)	
returns the channel name for the specified output	
<output-number>	the number of the output channel (zero based, i.e. the first output is 0) if -1 is applied, the default output (as specified in the instrument header) will be used

Examples

```
on init
  declare $count
  declare ui_menu $menu
  add_menu_item($menu,"Default",-1)

  $count := 0
  while($count < $NUM_OUTPUT_CHANNELS)
        add_menu_item($menu,output_channel_name($count),$count)
        inc($count)
  end while

  $menu := get_engine_par($ENGINE_PAR_OUTPUT_CHANNEL,0,-1,-1)
end on

on ui_control ($menu)
  set_engine_par($ENGINE_PAR_OUTPUT_CHANNEL,$menu,0,-1,-1)
end on
```

mirroring the output channel assignment menu of the first group

See Also

$NUM_OUTPUT_CHANNELS
$ENGINE_PAR_OUTPUT_CHANNEL

play_note()

```
play_note(<note-number>,<velocity>,<sample-offset>,<duration>)
```

generate a MIDI note, i.e. generate a note on message followed by a note off message

<note-number>	the note number to be generated (0 - 127)
<velocity>	velocity of the generated note (1 - 127)
<sample-offset>	this parameter specifies an offset in the sample in microseconds
<duration>	specifies the length of the generated note in microseconds this parameter also accepts two special values: -1: releasing the note which started the callback stops the sample 0: the entire sample is played

Remarks

- In DFD mode, the sample offset is dependent on the S. Mod value of the respective zones. Sample offset value greater than the zone's S.Mod setting are clipped to this value.
- You can retrieve the event ID of the played note by writing:
  `<variable> := play_note(<note>,<velocity>,<sample-offset>,<duration>)`

Examples

```
on note
  play_note($EVENT_NOTE+12,$EVENT_VELOCITY,0,-1)
end on
```

harmonizes the played note with the upper octave

```
on init
  declare $new_id
end on
on controller
  if ($CC_NUM = 64)
        if (%CC[64] = 127)
              $new_id := play_note(60,100,0,0)
        else
              note_off($new_id)
        end if
  end if
end on
```

trigger a MIDI note by pressing the sustain pedal

See Also

`note_off()`

random()

```
random(<min>,<max>)
```

generate a random number

Examples

```
on init
  declare $rnd_amt
  declare $new_vel
end on

on note
  $rnd_amt := $EVENT_VELOCITY * 10/100

  $new_vel := random($EVENT_VELOCITY - $rnd_amt,...
  $EVENT_VELOCITY + $rnd_amt)

  change_velo ($EVENT_ID,$new_vel)
end on
```

randomly changing velocities in by ±10 percent

See Also

read_persistent_var()

```
read_persistent_var(<variable>)
```

instantly reloads the value of a variable that was saved via `make_persistent`

Remarks

- This command can only be used within the init callback.
- The state of the variable is saved not only with the patch (or multi or host chunk), but also when a script is saved as a Kontakt preset (.nkp file).
- When replacing script code by copy and replacing the text, the values of persistent variables is also retained. Sometimes, when working on more complex scripts, you'll want to "flush" the variables by resetting the script, that is applying an empty script in the respective slot.

Examples

```
on init
  declare ui_label $label (1,1)
  declare ui_button $button
  set_text ($button,"$a := 10000")

  declare $a
  make_persistent ($a)
  {read_persistent_var ($a)}
  set_text ($label,$a)
end on

on ui_control ($button)
  $a := 10000
  set_text ($label,$a)
end on
```

after applying this script, click on the button and then save and close the NKI. After reloading it, the label will display 0 because the value of $a is initialized at the very end of the init callback. Now remove the {} around read_persistent_var and apply the script again. Voila.

See Also

`make_persistent()`

reset_ksp_timer

reset_ksp_timer
resets the KSP timer (`$KSP_TIMER`) to zero

Remarks

- Since the built-in variable `$KSP_TIMER` returns the engine uptime in microseconds (instead of milliseconds as with `$ENGINE_UPTIME`), the variable `$KSP_TIMER` will reach its limit after about 30 minutes due to its 32 bit nature. By using `reset_ksp_timer`, the variable is reset to 0.
- The main reason to use $KSP_TIMER is for debugging and optimization. It is a great tool to measure the efficiency of certain script passages.

Examples

```
on init
  declare $a
  declare $b
  declare $c
end on
on note
  reset_ksp_timer
  $c := 0
  while($c < 128)
        $a := 0
        while($a < 128)
              set_event_par...
              ($EVENT_ID,$EVENT_PAR_TUNE,random(-1000,1000))
              inc($a)
              end while
        inc($c)
        end while
  message($KSP_TIMER)
end on
```

a nested while loop – takes about 5400 to 5800 microseconds

See Also

`$ENGINE_UPTIME`
`$KSP_TIMER`

save_midi_file()

save_midi_file(<path>)	
saves a MIDI file	
<path>	the absolute path of the MIDI file.

Remarks

- Loading a new MIDI file will overwrite all changes made to a previously loaded MIDI file without any warning. Please make sure to save any changes.
- As this command needs an absolute path and MIDI files are not stored within the Resource Container, you need to use a command like `get_folder ($GET_FOLDER_PATCH_DIR)` or `get_folder ($GET_FOLDER_LIBRARY_DIR)` to load your file correctly.
- This command returns a unique value after is has finished its action. Please use it in combination with `$NI_ASYNC_ID` within the `on_async_complete` callback to avoid synchronization issues.

Example

```
on init
  declare $sync_id
  declare ui_button $Save
end on

on async_complete
  if ($NI_ASYNC_ID = $sync_id)
        $sync_id := -1
  end if
end on

on ui_control ($Save)
  $sync_id := save_midi_file (get_folder ($GET_FOLDER_PATCH_DIR)...
  & "midifile.mid")
  while ($sync_id # -1)
        wait (1)
  end while
end on
```

Saving a MIDI file.

See Also

`$NI_ASYNC_ID`
`load_midi_file()`
`on async_complete`

set_controller()

set_controller(<controller>,<value>)	
send a MIDI CC, pitchbend or channel pressure value	
<controller>	this parameter sets the type and in the case of MIDI CCs the CC number: • a number from 0 to 127 designates a MIDI CC number • $VCC_PITCH_BEND indicates Pitchbend • $VCC_MONO_AT indicates Channel Pressure (monophonic aftertouch)
<value>	the value of the specified controller MIDI CC and channel pressure values go from 0 to 127 PitchBend values go from -8192 to 8191

Remarks

- set_controller() should not be used within an init callback.

Examples

```
on note
  if ($EVENT_NOTE = 36)
        ignore_event($EVENT_ID)
        set_controller($VCC_MONO_AT,$EVENT_VELOCITY)
  end if
end on
on release
  if ($EVENT_NOTE = 36)
        ignore_event($EVENT_ID)
        set_controller($VCC_MONO_AT,0)
  end if
end on
```

Got a keyboard with no aftertouch? Press C1 instead.

See Also

ignore_controller
$VCC_PITCH_BEND
$VCC_MONO_AT

set_listener()

set_listener(<signal_type>,<parameter>)	
This command defines the signals the on listener callback should react to. You can only use it in the init callback.	
<signal_type>	the event on which the on listener callback should react. The following types are available: $NI_SIGNAL_TRANSP_STOP $NI_SIGNAL_TRANSP_START $NI_SIGNAL_TIMER_MS $NI_SIGNAL_TIMER_BEAT
<parameter>	the callback reacts (**1**) or doesn't react (**0**) to the event. When used with $NI_SIGNAL_TIMER_MS it defines that the on listener callback is triggered every <parameter> microseconds. When used with $NI_SIGNAL_TIMER_BEAT it defines how many times per beat the callback is triggered.

Examples

```
on init
  declare $mscount
  declare $beatcount

  set_listener ($NI_SIGNAL_TRANSP_STOP,1)
  set_listener ($NI_SIGNAL_TRANSP_START,1)
  set_listener ($NI_SIGNAL_TIMER_MS,1000000)
  set_listener ($NI_SIGNAL_TIMER_BEAT,1)
end on

on listener
  select ($NI_SIGNAL_TYPE)
        case $NI_SIGNAL_TRANSP_STOP
              message ("Playback was stopped.")
        case $NI_SIGNAL_TRANSP_START
              message ("Playback was started.")
        case $NI_SIGNAL_TIMER_MS
              inc ($mscount)
              message ($mscount & "sec. / " & $beatcount & " beats")
        case $NI_SIGNAL_TIMER_BEAT
              inc ($beatcount)
              message ($mscount & "sec. / " & $beatcount & " beats")
  end select
end on
```

a timer and a bpm counter

See Also

change_listener_par
$NI_SIGNAL_TYPE

set_rpn()/set_nrpn

set_rpn(<address>,<value>)	
send a rpn or nrpn message	
<address>	the rpn or nrpn address (0 - 16383)
<value>	the value of the rpn or nrpn message (0 - 16383)

Remarks

- Currently, Kontakt cannot handle rpn on nrpn messages as external modulation sources. You can however use these message for simple inter-script communication.

See Also

on rpn/nrpn
set_controller
$RPN_ADDRESS
$RPN_VALUE
msb()/lsb()

set_event_mark()

set_event_mark(<ID-number>,<bit-mark>	
assign the specified event to a specific event group	
<ID-number> <bit-mark>	the ID number of the event to be grouped here you can enter one of 28 marks from $MARK_1 to $MARK_28 which is addressed to the event. You can also address more than one mark to a single event, either by typing the command or by using the operator +.

Remarks

When dealing with commands that deal with event IDs, you can group events by using by_marks(<bit-mark>) instead of the individual ID, since the program needs to know that you want to address marks and not IDs.

Examples

```
on init
  declare $new_id
end on

on note
  set_event_mark($EVENT_ID,$MARK_1)

  $new_id := play_note($EVENT_NOTE + 12,120,0,-1)
  set_event_mark($new_id,$MARK_1 + $MARK_2)

  change_pan(by_marks($MARK_1),-1000,1) {both notes panned to left}
  change_pan(by_marks($MARK_2), 2000,1) {new note panned to right}
end on
```

the played note belongs to group 1, the harmonized belongs to group 1 and group 2

See Also

by_marks()
delete_event_mark()
$EVENT_ID
$ALL_EVENTS
$MARK_1 … $MARK_28

set_event_par()

set_event_par(<ID-number>,<parameter>,<value>)	
assign a parameter to a specific event	
<ID-number>	the ID number of the event
<parameter>	the event parameter, either one of four freely assignable event parameter: $EVENT_PAR_0 $EVENT_PAR_1 $EVENT_PAR_2 $EVENT_PAR_3 or the "built-in" parameters of a note event: $EVENT_PAR_VOLUME $EVENT_PAR_PAN $EVENT_PAR_TUNE $EVENT_PAR_NOTE $EVENT_PAR_VELOCITY
<value>	the value of the event parameter

Remarks

A note event always "carries" certain information like the note number, the played velocity, but also Volume, Pan and Tune. With `set_event_par()`, you can set either these parameters or use the freely assignable parameters like `$EVENT_PAR_0`. This is especially useful when chaining scripts, i.e. set an event parameter for an event in slot 1, then retrieve this information in slot 2 by using `get_event_par()`.
The event parameters are not influenced by the system scripts anymore.

Examples

```
on note
  set_event_par($EVENT_ID,$EVENT_PAR_NOTE,60)
end on
```

setting all notes to middle C3, same as change_note($EVENT_ID,60)

See Also

`get_event_par()`
`ignore_event()`
`set_event_par_arr()`
`get_event_par_arr()`

set_event_par_arr()

set_event_par_arr(<ID-number>,<parameter>,<value>,<groupindex>)	
special form of set_event_par(), used to set the group allow state of the specified event	
<ID-number>	the ID number of the note event
<parameter>	in this case, only $EVENT_PAR_ALLOW_GROUP
<value>	If set to **1**, the group set with <group-index> will be allowed for the event. If set to **0**, the group set with <group-index> will be disallowed for the event.
<group-index>	the index of the group for changing the specified note's group allow state

Remarks

- set_event_par_arr() is a special form (or to be more precise, it's the array variant) of set_event_par(). It is used to set the allow state of a specific event.

Examples

```
on note
  if (get_event_par_arr($EVENT_ID,$EVENT_PAR_ALLOW_GROUP,0) = 0)
        set_event_par_arr($EVENT_ID,$EVENT_PAR_ALLOW_GROUP,1,0)
  end if
end on
```

making sure, that the first group is always played

See Also

get_event_par_arr()
set_event_par()
$EVENT_PAR_ALLOW_GROUP

stop_wait()

stop_wait(<callbackID>,<par>)	
stops wait commands in a callback	
<callbackID> <par>	the callback's ID number in which the wait commands shall be stopped 0: stops only the current wait 1: stops the current wait and ignores all following wait commands in this callback.

Remarks

- Please be careful with while loops when you decide to ignore all wait commands in a callback!

See Also

```
wait()
wait_ticks()
```

ticks_to_ms()

```
ticks_to_ms(<ticks>)
```

converts a ticks value into microseconds

Examples

```
on init
  declare ui_label $songpos(1,1)
end on

on note
  while ($NOTE_HELD = 1)
        set_text ($songpos,ticks_to_ms($NI_SONG_POSITION)/1000000)
        wait(1000)
  end while
end on
```

displaying the song position in seconds

See Also

ms_to_ticks()
$NI_SONG_POSITION

wait()

wait(<wait-time>)
pauses the callback for the specified time in microseconds

Remarks

`wait()` stops the callback at the position in the script for the specified time. In other words, it freezes the callback (although other callbacks can be accessed or processed). After the specified time period the callback continues.

Examples

```
on note
  ignore_event($EVENT_ID)
  wait($DURATION_BAR - $DISTANCE_BAR_START)
  play_note($EVENT_NOTE,$EVENT_VELOCITY,0,-1)
end on
```

quantize all notes to the downbeat of the next measure

See Also

`stop_wait()`
`wait_ticks()`
`while()`
`$DURATION_QUARTER`

wait_ticks()

```
wait_ticks(<wait-time>)
```

pauses the callback for the specified time in ticks.

Remarks

`wait_ticks()` stops the callback at the position in the script for the specified time. In other words, it freezes the callback (although other callbacks can be accessed or processed). After the specified time period the callback continues.

See Also

`stop_wait()`
`wait()`

Built-in Variables & Constants

$ALL_GROUPS

addresses all groups in a `disallow_group()` and `allow_group()` function

$ALL_EVENTS

addresses all events in functions which deal with a event ID number

Bit Mark Constants

bit mark of an event group, to be used with `by_marks()`

```
$MARK_1
$MARK_2
…
$MARK_28
```

Callback Type Variables and Constants

`$NI_CALLBACK_ID`

returns a callback's ID. Every callback now gets its own ID number and it remains the same even within a function.

`$NI_CALLBACK_TYPE`

returns a certain number corresponding to the type of callback. This is especially useful when used in a function that is called from different callbacks.
The following constants are available:

```
$NI_CB_TYPE_NOTE
$NI_CB_TYPE_RELEASE
$NI_CB_TYPE_CONTROLLER
$NI_CB_TYPE_POLY_AT
$NI_CB_TYPE_INIT
$NI_CB_TYPE_UI_CONTROL
$NI_CB_TYPE_UI_UPDATE
$NI_CB_TYPE_PGS
$NI_CB_TYPE_MIDI_IN
$NI_CB_TYPE_RPN
$NI_CB_TYPE_NRPN
$NI_CB_TYPE_LISTENER
$NI_CB_TYPE_ASYNC_OUT
```

%CC[<controller-number>]

current controller value for the specified controller.

$CC_NUM

controller number of the controller which triggered the callback

%CC_TOUCHED[<controller-number>]

1 if the specified controller value has changed, 0 otherwise

$CURRENT_SCRIPT_SLOT

the script slot of the current script (zero based, i.e. the first script slot is 0)

$DISTANCE_BAR_START

returns the time of a note on message in µsec from the beginning of the current bar with respect to the current tempo

$DURATION_BAR

returns the duration in µsec of one bar with respect to the current tempo.

This variable only works if the clock is running, otherwise it will return a value of zero.
You can also retrieve the duration of one bar bys using $SIGNATURE_NUM and $SIGNATURE_DENOM in combination with $DURATION_QUARTER.

$DURATION_QUARTER

duration of a quarter note in microseconds, with respect to the current tempo.

Also available:
$DURATION_EIGHTH
$DURATION_SIXTEENTH
$DURATION_QUARTER_TRIPLET
$DURATION_EIGHTH_TRIPLET
$DURATION_SIXTEENTH_TRIPLET

$ENGINE_UPTIME

Returns the time period in milliseconds (not microseconds) that has passed since the start of Kontakt

$EVENT_ID

unique ID number of the event which triggered the callback

$EVENT_NOTE

note number of the event which triggered the callback

$EVENT_VELOCITY

velocity of the note which triggered the callback

Event Parameter Constants

event parameters to be used with `set_event_par()` and `get_event_par()`
`$EVENT_PAR_0`
`$EVENT_PAR_1`
`$EVENT_PAR_2`
`$EVENT_PAR_3`
`$EVENT_PAR_VOLUME`
`$EVENT_PAR_PAN`
`$EVENT_PAR_TUNE`
`$EVENT_PAR_NOTE`
`$EVENT_PAR_VELOCITY`

To be used with `set_event_par_arr()`:
`$EVENT_PAR_ALLOW_GROUP`

To be used with `get_event_par()`:
`$EVENT_PAR_SOURCE` (-1 if event originates from outside, otherwise slot number 0 - 4)
`$EVENT_PAR_PLAY_POS` (returns the value of the play cursor within a zone)

Event Status Constants

`$EVENT_STATUS_INACTIVE`
`$EVENT_STATUS_NOTE_QUEUE`
`$EVENT_STATUS_MIDI_QUEUE`

%GROUPS_AFFECTED

an array with the group indices of those groups that are affected by the current Note On or Note Off events

%GROUPS_SELECTED

an array with each array index pointing to the group with the same index.

If a group is selected for editing the corresponding array index contains a 1, otherwise 0

Hide Part Constants

to be used with `hide_part()`

`$HIDE_PART_BG` {Background of knobs, labels, value edits and tables}
`$HIDE_PART_VALUE` {value of knobs}
`$HIDE_PART_TITLE` {title of knobs}
`$HIDE_PART_MOD_LIGHT` {mod ring light of knobs}
`$HIDE_PART_NOTHING` {Show all}
`$HIDE_WHOLE_CONTROL`

%KEY_DOWN[<note-number>]

array which contains the current state of all keys. 1 if the key is held, 0 otherwise

%KEY_DOWN_OCT[<note-number>]

1 if a note independently of the octave is held, 0 otherwise

Knob Unit Mark Constants

to be used with `set_knob_unit()`

```
$KNOB_UNIT_NONE
$KNOB_UNIT_DB
$KNOB_UNIT_HZ
$KNOB_UNIT_PERCENT
$KNOB_UNIT_MS
$KNOB_UNIT_ST
$KNOB_UNIT_OCT
```

$KSP_TIMER

Returns the time period in microseconds that has passed since the start of Kontakt.
Can be reset with `reset_ksp_timer`

$NI_ASYNC_EXIT_STATUS

can be used in the `on async_complete` callback to determine if the command that triggered the callback has successfully completed its action.

$NI_ASYNC_ID

can be used in the `on async_complete` callback to determine if a command (like `load_ir_sample`, `save_midi_file` etc.) has finished its action.

$NI_SIGNAL_TYPE

can be used in the on listener callback to determine which signal type triggered the callback.

$NI_SONG_POSITION

Returns the host's current song position in 960 ticks per quarter note. In KONTAKT 4's stand-alone mode there is no real transport - the playback is started with KONTAKT 4's engine start.

Two converter functions are available:

```
<ticks> := ms_to_ticks(<microseconds>)
<microseconds> := ticks_to_ms(<ticks>)
```

It is very important to know that `$NI_SONG_POSITION` works in real time and therefore it is crucial to be careful when working with it. It is possible that in some special situations (like tempo changes within the host), ticks are left out. As an example, a modulo division to receive the trigger for a special timing (like offbeats) could return wrong results. So you always should use a certain tolerance and re-check every now in then that you are still in sync and that you still get the results you would expect from your script.

%NOTE_DURATION[<note-number>]

note length since note-start in µsec for each key

$NOTE_HELD

1 if the key which triggered the callback is still held, **0** otherwise

$NUM_GROUPS

total amount of groups in an instrument

$NUM_OUTPUT_CHANNELS
total amount of output channels of the respective Kontakt Multi (not counting Aux channels)

$NUM_ZONES
total amount of zones in an instrument

$PLAYED_VOICES_INST
the amount of played voices of the respective instrument

$PLAYED_VOICES_TOTAL
the amount of played voices all instruments

%POLY_AT[<note-number>]
the polyphonic aftertouch value of the specified note number

$POLY_AT_NUM
the note number of the polyphonic aftertouch note which triggered the callback

$REF_GROUP_IDX
group index number of the currently viewed group

$RPN_ADDRESS
the parameter number of a received rpn/nrpn message (0 – 16383)

$RPN_VALUE
the value of a received rpn or nrpn message (0 – 16383)

$SIGNATURE_NUM
numerator of the current time signature, i.e. **4**/4

$SIGNATURE_DENOM
denominator of the current time signature, i.e. 4/**4**

$VCC_MONO_AT
the value of the virtual cc controller for mono aftertouch (channel pressure)

$VCC_PITCH_BEND
the value of the virtual cc controller for pitch bend

Waveform Flag Constants	
to be used with `attach_zone()`	
$UI_WAVEFORM_USE_SLICES	shows(**1**) or hides(**0**) the zone's slices within the UI waveform
$UI_WAVEFORM_USE_TABLE	shows(**1**) or hides(**0**) a table for each slice
$UI_WAVEFORM_TABLE_IS_BIPOLAR	toggles between bipolar(**1**) and unipolar(**0**) mode
$UI_WAVEFORM_USE_MIDI_DRAG	shows(**1**) or hides(**0**) a little midi drag icon in the top right corner to allow midi drag & drop into a host software

Waveform Property Constants	
to be used with `get/set_ui_wf_property()`	
`$UI_WF_PROP_PLAY_CURSOR`	shows(1) or hides(0) a play cursor within the UI waveform
`$UI_WF_PROP_FLAGS`	returns the value of the indexed slice's flags (get_ui_wf_property only)
`$UI_WF_PROP_TABLE_VAL`	sets or returns the value of the indexed slice's table
`$UI_WF_PROP_TABLE_IDX_HIGHLIGHT`	highlights the indexed slice within the UI waveform
`$UI_WF_PROP_MIDI_DRAG_START_NOTE`	defines the start note for the midi drag & drop function

Control Parameter Variables

```
$CONTROL_PAR_NONE
```
nothing will be applied to the control

```
$CONTROL_PAR_POS_X
```
sets the horizontal position in pixel

```
$CONTROL_PAR_POS_Y
```
sets the vertical position in pixel

```
$CONTROL_PAR_GRID_X
```
sets the horizontal position in grids

```
$CONTROL_PAR_GRID_Y
```
sets the vertical position in grids

```
$CONTROL_PAR_WIDTH
```
sets the width of the control in pixel

```
$CONTROL_PAR_HEIGHT
```
sets the height of the control in pixel

```
$CONTROL_PAR_GRID_WIDTH
```
sets the width of the control in grids

```
$CONTROL_PAR_GRID_HEIGHT
```
sets the height of the control in grids

```
$CONTROL_PAR_HIDE
```
sets the hide status

```
$CONTROL_PAR_MIN_VALUE
```
sets the minimum value

```
$CONTROL_PAR_MAX_VALUE
```
sets the maximum value

```
$CONTROL_PAR_VALUE
```
sets the value

```
$CONTROL_PAR_DEFAULT_VALUE
```
sets the default value

`$CONTROL_PAR_HELP`

sets the help text which is displayed in the info pane when hovering the control

`$CONTROL_PAR_PICTURE`

sets the picture name

`$CONTROL_PAR_TEXT`

sets the control text, similar to `set_text()`

`$CONTROL_PAR_TEXTLINE`

adds a text line, similar to `add_text_line()`

`$CONTROL_PAR_LABEL`

sets the knob label, similar to `set_knob_label()`

`$CONTROL_PAR_UNIT`

sets the knob unit, similar to `set_knob_unit()`

`$CONTROL_PAR_MOUSE_BEHAVIOUR`

a value from -5000 to 5000, setting the move direction of a slider and its drag-scale

`$CONTROL_PAR_PICTURE_STATE`

the picture state of the control for tables, value edits and labels

`$CONTROL_PAR_FONT_TYPE`

sets the font type.
Only Kontakt 4 factory fonts can be used, the font itself is designated by a number (currently 0 to 24)

`$CONTROL_PAR_TEXTPOS_Y`

shifts the vertical position in pixels of text in buttons, menus, switches and labels

`$CONTROL_PAR_TEXT_ALIGNMENT`

the text alignment in buttons, menus, switches and labels.

The following values can be used:
0: left
1: centered
2: right

`$CONTROL_PAR_SHOW_ARROWS`

hide arrows of value edits:

0: arrows are hidden
1: arrows are shown

`$CONTROL_PAR_BAR_COLOR`

sets the color of the step bar in UI tables and UI waveforms

`$CONTROL_PAR_ZERO_LINE_COLOR`

sets the color of the middle line in UI tables

`$CONTROL_PAR_AUTOMATION_NAME`

assigns an automation name to a UI control when used with `set_control_par_str()`

`$CONTROL_PAR_ALLOW_AUTOMATION`

defines if an ui_control can be automated (**1**) or not (**0**). By default automation is enabled. You can use this control_par only in the init callback.

`$CONTROL_PAR_KEY_SHIFT`

returns **1** when the shift key was pressed (**0** otherwise) when an UI control was last touched. Menus and value edits are not supported.

The basic shift modifier functionality on sliders and knobs is preserved.

`$CONTROL_PAR_KEY_ALT`

returns **1** when the ALT key was pressed (**0** otherwise) when an UI control was last touched to be used with `get_control_par()`, menus and value edits are not supported

`$CONTROL_PAR_KEY_CONTROL`

returns **1** when the Control key was pressed (**0** otherwise) when an UI control was last touched to be used with `get_control_par()`, menus and value edits are not supported

`$CONTROL_PAR_NUM_ITEMS`

returns the number of menu entries of a specific dropdown menu. This control parameter only works with `get_control_par()`.

`$CONTROL_PAR_SELECTED_ITEM_IDX`

returns the index of the currently selected menu entry. This control parameter only works with `get_control_par()`.

`$CONTROL_PAR_BG_COLOR`

sets the background color of the UI level meter

`$CONTROL_PAR_OFF_COLOR`

sets the second background color of the UI level meter

`$CONTROL_PAR_ON_COLOR`

sets the main level meter color of the UI level meter

`$CONTROL_PAR_OVERLOAD_COLOR`

sets the color of the level meter's overload section

$CONTROL_PAR_PEAK_COLOR

sets the color of the little bar showing the current peak level

$CONTROL_PAR_VERTICAL

aligns a UI level meter vertically (**1**) or horizontally (**0,default**)

$CONTROL_PAR_BASEPATH

sets the basepath of the UI file browser. This control par can only be used in the init callback. Be careful with the number of subfolders of the basepath as it might take too long to scan the sub file system. The scan process takes place every time the NKI is loaded.

$CONTROL_PAR_COLUMN_WIDTH

sets the width of the browser columns. This control par can only be used in the init callback.

$CONTROL_PAR_FILEPATH

sets the actual path of the UI file browser which must be a subpath of the basepath. This control par is useful for recalling the last status of the browser upon loading the instrument. This control par can only be used in the init callback.

$CONTROL_PAR_FILE_TYPE

sets the file type that the file selector works for. This control par can only be used in the init callback.

- $NI_FILE_TYPE_MIDI
- $NI_FILE_TYPE_AUDIO
- $NI_FILE_TYPE_ARRAY

$INST_ICON_ID

the (fixed) ID of the instrument icon.

It's possible to hide the instrument icon:

```
set_control_par($INST_ICON_ID,$CONTROL_PAR_HIDE,$HIDE_WHOLE_CONTROL)
```

It's also possible to load a different picture file for the instrument icon:

```
set_control_par_str($INST_ICON_ID,$CONTROL_PAR_PICTURE,<file-name>)
```

$INST_WALLPAPER_ID

The (fixed) ID of the instrument wallpaper. It is used in a similar way as $INST_ICON_ID:

```
set_control_par_str ($INST_WALLPAPER_ID,$CONTROL_PAR_PICTURE,<file_name>)
```

This command can only be used in the init callback. Note that a wallpaper set via script replaces the one set in the instrument options plus it won't be checked in the samples missing dialog when loading the wallpaper from a resource container. You should also be aware of the fact that this command only supports wallpapers that are located within the resource container. If you use it in different script slots then the last wallpaper set will be the one that is loaded.

Engine Parameter Commands

find_mod()

find_mod(<group-index>,<mod-name>)	
returns the slot index of an internal modulator or external modulation slot	
<group-index>	the index of the group
<mod-name>	the name of the modulator or modulation slot Each modulator or modulation slot has a predefined name, based on the modulation source and target. The name can be changed with the script editor's edit area open and right-clicking on the modulator or modulation slot.

Examples

```
on init
   declare $count
   declare ui slider $test (0,1000000)
   $test := get engine par($ENGINE PAR MOD TARGET INTENSITY,0,...
   find mod(0,"VEL VOLUME"),-1)
end on
on ui_control ($test)
   $count := 0
   while($count < $NUM GROUPS)
          set engine par($ENGINE PAR MOD TARGET INTENSITY,$test,$count,...
          find_mod($count,"VEL_VOLUME"),-1)
          inc($count)
   end while
end on
```

creating a slider which mirrors the velocity to volume modulation intensity of all groups

```
on init
   declare ui slider $Attack (0,1000000)
   set control par str(get ui id($Attack),$CONTROL PAR AUTOMATION NAME,"Attack")
   declare $vol env idx
   $vol_env_idx := find_mod(0,"VOL_ENV")
   declare const $grp_idx := 0
   $Attack := get engine par($ENGINE PAR ATTACK,$grp idx,$vol env idx,-1)
   set control par str(get ui id($Attack),$CONTROL PAR LABEL,...
   get engine par disp($ENGINE PAR ATTACK,$grp idx,$vol env idx,-1) & " ms")
end on
   on ui_control ($Attack)
   set engine par($ENGINE PAR ATTACK,$Attack,$grp idx,$vol env idx,-1)
   set control par str(get ui id($Attack),$CONTROL PAR LABEL,...
   get engine par disp($ENGINE PAR ATTACK,$grp idx,$vol env idx,-1) & " ms")
end on
```

controlling the attack time of the volume envelope of the first group. Note: the envelope has been manually renamed to "VOL_ENV". Also contains the readout of the attack knob when automating the control.

See Also

find_target()
set_engine_par()

find_target()

find_target(<group-index>,<mod-index>,<target-name>)	
returns the slot index of a modulation slot of an internal modulator	
<group-index>	the index of the group
<mod-index>	the slot index of the internal modulator. Can be retrieved with find_mod(<group-idx>,<mod-name>)
<target-name>	the name of the modulation slot Each mmodulation slot has a predefined name, based on the modulation source and target. The name can be changed with the script editor's edit area open and right-clicking on the modulation slot.

Examples

```
on init
   declare ui knob $Knob (-100,100,1)
   declare $mod idx
   $mod idx := find mod(0,"FILTER ENV")

   declare $target_idx
   $target idx := find target(0,$mod idx,"ENV AHDSR CUTOFF")
end on

on ui control ($Knob)
   if ($Knob < 0)
          set engine par ($MOD TARGET INVERT SOURCE,...
          1,0,$mod idx,$target idx)
   else
          set engine par ($MOD TARGET INVERT SOURCE,...
          0,0,$mod_idx,$target_idx)
   end if
   set engine par($ENGINE PAR MOD TARGET INTENSITY,...
   abs($Knob*10000),0,$mod idx,$target idx)
end on
```

controlling the filter envelope amount of an envelope to filter cutoff modulation in the first group. Note: the the filter envelope has been manually renamed to "FILTER_ENV"

See Also

find_mod()
set_engine_par()

get_engine_par()

get_engine_par(<parameter>,<group>,<slot>,<generic>)	
return the value of a specific engine parameter	
<parameter>	specifies the engine parameter
<group>	the index (zero based) of the group in which the specified parameter resides. If the specified parameter resides on an **Instrument** level, enter **-1**.
<slot>	the slot index (zero based) of the specified parameter (applies only to group/instrument effects, modulators and modulation intensities) For group/instrument effects, this parameter specifies the slot in which the effect resides (zero-based). For modulators and modulation intensities, this parameters specifies the index which you can retrieve by using: find_mod(<group-idx>,<mod-name>) For all other applications, set this parameter to **-1**.
<generic>	this parameter applies to instrument effects and to internal modulators. For instrument effects, this parameter distinguishes between 1: Insert Effect 0: Send Effect For internal modulators, this parameter specifies the modulation slider which you can retrieve by using find_target(<group-idx>,<mod-idx>,<target-name>) For all other applications, set this parameter to **-1**

Examples

```
on init
  declare $a

  declare ui_label $label (2,6)
  set_text ($label,"Release Trigger Groups:")

  while ($a < $NUM_GROUPS)
        if(get_engine_par($ENGINE_PAR_RELEASE_TRIGGER ,$a,-1,-1)=1)
          add_text_line($label,group_name($a)&" (Index: "&$a&")")
        end if
        inc($a)
  end while
end on
```

output the name and index of release trigger group

```
on init
   declare ui label $label (2,6)

   declare ui_button $Refresh

   declare !effect name[128]
   !effect name[$EFFECT TYPE NONE] := "None"
   !effect name[$EFFECT TYPE PHASER] := "Phaser"
   !effect name[$EFFECT_TYPE_CHORUS] := "Chorus"
   !effect_name[$EFFECT_TYPE_FLANGER] := "Flanger"
   !effect name[$EFFECT TYPE REVERB] := "Reverb"
   !effect name[$EFFECT TYPE DELAY] := "Delay"
   !effect name[$EFFECT TYPE IRC] := "Convolution"
   !effect name[$EFFECT TYPE GAINER] := "Gainer"

   declare $count
   while ($count < 8)
          add text line($label,"Slot: " & $count+1 & ": " & ...
          !effect name[get engine par($ENGINE PAR SEND EFFECT TYPE,-1,$count,-1)])
   inc($count)
   end while

end on

on ui control ($Refresh)
   set_text($label,"")
   $count := 0
   while ($count < 8)
          add text line($label,"Slot: " & $count+1 & ": " & ...
          !effect name[get engine par($ENGINE PAR SEND EFFECT TYPE,-1,$count,-1)])
   inc($count)
   end while

   $Refresh := 0
end on
```

output the effect types of all eight slots of send effects

See Also

Module Status Retrieval

get_engine_par_disp()

get_engine_par_disp(<parameter>,<group>,<slot>,<generic>)	
return the displayed value of a specific engine parameter	
<parameter>	specifies the engine parameter
<group>	the index (zero based) of the group in which the specified parameter resides. If the specified parameter resides on an **Instrument** level, enter **-1**.
<slot>	the slot index (zero based) of the specified parameter (applies only to group/instrument effects, modulators and modulation intensities) For group/instrument effects, this parameter specifies the slot in which the effect resides (zero-based). For modulators and modulation intensities, this parameters specifies the index which you can retrieve by using: find_mod(<group-idx>,<mod-name>) For all other applications, set this parameter to **-1**.
<generic>	this parameter applies to instrument effects and to internal modulators. For instrument effects, this parameter distinguishes between 1: Insert Effect 0: Send Effect For internal modulators, this parameter specifies the modulation slider which you can retrieve by using find_target(<group-idx>,<mod-idx>,<target-name>) For all other applications, set this parameter to **-1**

Examples

```
on init
  declare $a

  declare ui_label $label (2,6)
  set_text ($label,"Group Volume Settings:")

  while ($a < $NUM_GROUPS)
        add_text_line($label,group_name($a) & ": "  & ...
        get_engine_par_disp($ENGINE_PAR_VOLUME,$a,-1,-1) & " dB")
        inc($a)
  end while
end on
```

query the group volume settings in an instrument

load_ir_sample()

load_ir_sample(<file-name>,<slot>,<generic>)	
load an impulse response sample into KONTAKT's convolution effect	
<file-name>	the file name of the sample, specified with a relative path If no path is specified and the instrument uses a Resource Container, the command will look for the specified sample within the “ir_samples” folder of the Resource Container. If there is no Resource Container available the folder "ir_samples" within the KONTAKT user folder will be checked. You can find it at these locations: **(OS X)** /Users/<username>/Documents/Native Instruments/Kontakt 4 **(Windows)** C:\Users\<username>\Documents\Native Instruments\Kontakt 4\ Please note that sub directories inside the "ir_samples" folder will not be scanned and it is not recommended to add them manually via text strings. Doing so could lead to problems because subfolders are being ignored during the creation of a Resource Container monolith.
<slot>	the slot index of the convolution effect (zero-based)
<generic>	specifies whether the convolution effect is used as an **1:** Insert Effect **0:** Send Effect

Remarks

- This command returns a unique value after is has finished its action. Please use it in combination with $NI_ASYNC_ID within the on_async_complete callback to avoid synchronization issues.

Examples

```
on init
 declare ui_button $Load
end on
on ui_control ($Load)
  load_ir_sample(get_folder($GET_FOLDER_FACTORY_DIR) & ...
  "presets/effects/convolution/<<<K4IR.nkx>>>/K4 IR Samples/Concert
Hall A.wav"...
  ,0,0)

  $Load := 0
end on
```

load a factory IR sample into a convolution send effect in the first slot

See Also

```
$NI_ASYNC_ID
get_folder()
on async_complete
```

set_engine_par()

set_engine_par(<parameter>,<value>,<group>,<slot>,<generic>)	
control automatable Kontakt parameters and bypass buttons	
<parameter>	the parameter to be controlled with a built-in variable like $ENGINE_PAR_CUTOFF
<value>	The value to which the specified parameter is set. The range of values is always 0 to 1000000.
<group>	the index (zero based) of the group in which the specified parameter resides. If the specified parameter resides on an **Instrument** level, enter **-1**. Busses also reside on **Instrument** level, so you need to set <group> to **-1** if you want to address a bus.
<slot>	the slot index (zero based) of the specified parameter (applies only to group/instrument effects, modulators and modulation intensities) For group/instrument effects, this parameter specifies the slot in which the effect resides (zero-based). For modulators and modulation intensities, this parameters specifies the index which you can retrieve by using: find_mod(<group-idx>,<mod-name>) For all other applications, set this parameter to **-1**.
<generic>	this parameter applies to instrument effects and to internal modulators. For instrument effects, this parameter distinguishes between **1:** Insert Effect **0:** Send Effect For busses, this parameter specifies the actual bus: $NI_BUS_OFFSET + [0-15] one of the 16 busses For internal modulators, this parameter specifies the modulation slider which you can retrieve by using find_target(<group-idx>,<mod-idx>,<target-name>) For all other applications, set this parameter to **-1**

Examples

```
on init
  declare ui_knob $Volume (0,1000000,1000000)
end on
on ui_control ($Volume)
  set_engine_par($ENGINE_PAR_VOLUME,$Volume,-1,-1,-1)
end on
```

controlling instrument volume

```
on init
  declare ui_knob $Freq (0,1000000,1000000)
  declare ui_button $Bypass
end on

on ui_control ($Freq)
  set_engine_par($ENGINE_PAR_CUTOFF,$Freq,0,0,-1)
end on

on ui_control ($Bypass)
  set_engine_par($ENGINE_PAR_EFFECT_BYPASS,$Bypass,0,0,-1)
end on
```

controlling the cutoff and Bypass button of any filter module in the first slot of the first group

```
on init
  declare ui_knob $Knob (-100,100,1)
  declare $mod_idx
  $mod_idx := find_mod(0,"FILTER_ENV")

  declare $target_idx
  $target_idx := find_target(0,$mod_idx,"ENV_AHDSR_CUTOFF")
end on

on ui_control ($Knob)
  if ($Knob < 0)
        set_engine_par ($MOD_TARGET_INVERT_SOURCE,...
        1,0,$mod_idx,$target_idx)
  else
        set_engine_par ($MOD_TARGET_INVERT_SOURCE,...
        0,0,$mod_idx,$target_idx)
  end if
  set_engine_par($ENGINE_PAR_MOD_TARGET_INTENSITY,...
  abs($Knob*10000),0,$mod_idx,$target_idx)
end on
```

controlling the filter envelope amount of an envelope to filter cutoff modulation in the first group. Note: the the filter envelope has been manually renamed to "FILTER_ENV"

```
on init
  declare ui_knob $Vol (-0,1000000,1)
end on

on ui_control ($Vol)
  set_engine_par($ENGINE_PAR_VOLUME,$Vol,-1,-1,$NI_BUS_OFFSET + 15)
end on
```

controlling the amplifier volume of the 16th bus

Engine Parameter Variables

Instrument, Source and Amp Module

$ENGINE_PAR_VOLUME

instrument volume

$ENGINE_PAR_PAN

instrument panorama

$ENGINE_PAR_TUNE

instrument tuning

Source Module

$ENGINE_PAR_TUNE
$ENGINE_PAR_SMOOTH
$ENGINE_PAR_FORMANT
$ENGINE_PAR_SPEED
$ENGINE_PAR_GRAIN_LENGTH
$ENGINE_PAR_SLICE_ATTACK
$ENGINE_PAR_SLICE_RELEASE
$ENGINE_PAR_TRANSIENT_SIZE

Amp Module

$ENGINE_PAR_VOLUME
$ENGINE_PAR_PAN

$ENGINE_PAR_OUTPUT_CHANNEL
The range of the $ENGINE_PAR_OUTPUT_CHANNEL setting has been adjusted for groups so that $NI_BUS_OFFSET + [0 – 15] would route a group's output to one of the busses.
The range of the $ENGINE_PAR_OUTPUT_CHANNEL setting has also been adjusted for busses so that -2 refers to the "program out (bypass insert FX)" option.

Filter and EQ

$ENGINE_PAR_CUTOFF

cutoff frequency of all filters

$ENGINE_PAR_RESONANCE

resonance of all filters

$ENGINE_PAR_EFFECT_BYPASS

bypass button of all filters/EQ

3x2 Versatile

$ENGINE_PAR_FILTER_SHIFTB
$ENGINE_PAR_FILTER_SHIFTC
$ENGINE_PAR_FILTER_RESB
$ENGINE_PAR_FILTER_RESC
$ENGINE_PAR_FILTER_TYPEA
$ENGINE_PAR_FILTER_TYPEB
$ENGINE_PAR_FILTER_TYPEC
$ENGINE_PAR_FILTER_BYPA
$ENGINE_PAR_FILTER_BYPB
$ENGINE_PAR_FILTER_BYPC
$ENGINE_PAR_FILTER_GAIN

EQ

$ENGINE_PAR_FREQ1
$ENGINE_PAR_BW1
$ENGINE_PAR_GAIN1
$ENGINE_PAR_FREQ2
$ENGINE_PAR_BW2
$ENGINE_PAR_GAIN2
$ENGINE_PAR_FREQ3
$ENGINE_PAR_BW3
$ENGINE_PAR_GAIN3

Solid G-EQ

$ENGINE_PAR_SEQ_LF_GAIN
$ENGINE_PAR_SEQ_LF_FREQ
$ENGINE_PAR_SEQ_LF_BELL
$ENGINE_PAR_SEQ_LMF_GAIN
$ENGINE_PAR_SEQ_LMF_FREQ
$ENGINE_PAR_SEQ_LMF_Q
$ENGINE_PAR_SEQ_HMF_GAIN
$ENGINE_PAR_SEQ_HMF_FREQ
$ENGINE_PAR_SEQ_HMF_Q
$ENGINE_PAR_SEQ_HF_GAIN
$ENGINE_PAR_SEQ_HF_FREQ
$ENGINE_PAR_SEQ_HF_BELL

Insert Effects

$ENGINE_PAR_EFFECT_BYPASS

bypass button of all insert effects

$ENGINE_PAR_INSERT_EFFECT_OUTPUT_GAIN

output gain of all insert effects

Compressor

```
$ENGINE_PAR_THRESHOLD
$ENGINE_PAR_RATIO
$ENGINE_PAR_COMP_ATTACK
$ENGINE_PAR_COMP_DECAY
```

Limiter

```
$ENGINE_PAR_LIM_IN_GAIN
$ENGINE_PAR_LIM_RELEASE
```

Surround Panner

```
$ENGINE_PAR_SP_OFFSET_DISTANCE
$ENGINE_PAR_SP_OFFSET_AZIMUTH
$ENGINE_PAR_SP_OFFSET_X
$ENGINE_PAR_SP_OFFSET_Y
$ENGINE_PAR_SP_LFE_VOLUME
$ENGINE_PAR_SP_SIZE
$ENGINE_PAR_SP_DIVERGENCE
```

Saturation

```
$ENGINE_PAR_SHAPE
```

Lo-Fi

```
$ENGINE_PAR_BITS
$ENGINE_PAR_FREQUENCY
$ENGINE_PAR_NOISELEVEL
$ENGINE_PAR_NOISECOLOR
```

Stereo Modeller

```
$ENGINE_PAR_STEREO
$ENGINE_PAR_STEREO_PAN
```

Distortion

```
$ENGINE_PAR_DRIVE
$ENGINE_PAR_DAMPING
```

Send Levels

```
$ENGINE_PAR_SENDLEVEL_0
$ENGINE_PAR_SENDLEVEL_1
$ENGINE_PAR_SENDLEVEL_2
<...>
$ENGINE_PAR_SENDLEVEL_7
```

Skreamer

```
$ENGINE_PAR_SK_TONE
$ENGINE_PAR_SK_DRIVE
$ENGINE_PAR_SK_BASS
$ENGINE_PAR_SK_BRIGHT
$ENGINE_PAR_SK_MIX
```

Rotator

```
$ENGINE_PAR_RT_SPEED
$ENGINE_PAR_RT_BALANCE
$ENGINE_PAR_RT_ACCEL_HI
$ENGINE_PAR_RT_ACCEL_LO
$ENGINE_PAR_RT_DISTANCE
$ENGINE_PAR_RT_MIX
```

Twang

```
$ENGINE_PAR_TW_VOLUME
$ENGINE_PAR_TW_TREBLE
$ENGINE_PAR_TW_MID
$ENGINE_PAR_TW_BASS
```

Cabinet

```
$ENGINE_PAR_CB_SIZE
$ENGINE_PAR_CB_AIR
$ENGINE_PAR_CB_TREBLE
$ENGINE_PAR_CB_BASS

$ENGINE_PAR_CABINET_TYPE
```

AET Filter Module

```
$ENGINE_PAR_EXP_FILTER_MORPH
$ENGINE_PAR_EXP_FILTER_AMOUNT
```

Tape Saturator

```
$ENGINE_PAR_TP_GAIN
$ENGINE_PAR_TP_WARMTH
$ENGINE_PAR_TP_HF_ROLLOFF
```

Transient Master

```
$ENGINE_PAR_TR_INPUT
$ENGINE_PAR_TR_ATTACK
$ENGINE_PAR_TR_SUSTAIN
```

Solid Bus Comp

```
$ENGINE_PAR_SCOMP_THRESHOLD
$ENGINE_PAR_SCOMP_RATIO
$ENGINE_PAR_SCOMP_ATTACK
$ENGINE_PAR_SCOMP_RELEASE
$ENGINE_PAR_SCOMP_MAKEUP
$ENGINE_PAR_SCOMP_MIX
```

Send Effects

$ENGINE_PAR_SEND_EFFECT_BYPASS

bypass button of all send effects

$ENGINE_PAR_SEND_EFFECT_DRY_LEVEL

dry amount of send effects when used in an insert chain

$ENGINE_PAR_SEND_EFFECT_OUTPUT_GAIN

when used with send effects, this controls either:
- **wet** amount of send effects when used in an **insert** chain
- **return** amount of send effects when used in a **send** chain

Phaser

$ENGINE_PAR_PH_DEPTH
$ENGINE_PAR_PH_SPEED
$ENGINE_PAR_PH_PHASE
$ENGINE_PAR_PH_FEEDBACK

Flanger

$ENGINE_PAR_FL_DEPTH
$ENGINE_PAR_FL_SPEED
$ENGINE_PAR_FL_PHASE
$ENGINE_PAR_FL_FEEDBACK
$ENGINE_PAR_FL_COLOR

Chorus

$ENGINE_PAR_CH_DEPTH
$ENGINE_PAR_CH_SPEED
$ENGINE_PAR_CH_PHASE

Reverb

$ENGINE_PAR_RV_PREDELAY
$ENGINE_PAR_RV_SIZE
$ENGINE_PAR_RV_COLOUR
$ENGINE_PAR_RV_STEREO
$ENGINE_PAR_RV_DAMPING

Delay

$ENGINE_PAR_DL_TIME
$ENGINE_PAR_DL_DAMPING
$ENGINE_PAR_DL_PAN
$ENGINE_PAR_DL_FEEDBACK

Convolution

$ENGINE_PAR_IRC_PREDELAY
$ENGINE_PAR_IRC_LENGTH_RATIO_ER
$ENGINE_PAR_IRC_FREQ_LOWPASS_ER
$ENGINE_PAR_IRC_FREQ_HIGHPASS_ER
$ENGINE_PAR_IRC_LENGTH_RATIO_LR
$ENGINE_PAR_IRC_FREQ_LOWPASS_LR
$ENGINE_PAR_IRC_FREQ_HIGHPASS_LR

Gainer

$ENGINE_PAR_GN_GAIN

Modulation

$ENGINE_PAR_MOD_TARGET_INTENSITY

the intensity slider of a modulation assignment, controls the modulation amount

$MOD_TARGET_INVERT_SOURCE

the Invert button of a modulation assignment, inverts the modulation amount

$ENGINE_PAR_INTMOD_BYPASS

the bypass button of an internal modulator (e.g. AHDSR envelope, LFO)

AHDSR

```
$ENGINE_PAR_ATK_CURVE
$ENGINE_PAR_ATTACK
$ENGINE_PAR_HOLD
$ENGINE_PAR_DECAY
$ENGINE_PAR_SUSTAIN
$ENGINE_PAR_RELEASE
```

DBD

```
$ENGINE_PAR_DECAY1
$ENGINE_PAR_BREAK
$ENGINE_PAR_DECAY2
```

LFO

For all LFOs:

```
$ENGINE_PAR_INTMOD_FREQUENCY
$ENGINE_PAR_LFO_DELAY
```

For Rectangle:

```
$ENGINE_PAR_INTMOD_PULSEWIDTH
```

For Multi:

```
$ENGINE_PAR_LFO_SINE
$ENGINE_PAR_LFO_RECT
$ENGINE_PAR_LFO_TRI
$ENGINE_PAR_LFO_SAW
$ENGINE_PAR_LFO_RAND
```

Glide

```
$ENGINE_PAR_GLIDE_COEF
```

Module Status Retrieval

$ENGINE_PAR_EFFECT_TYPE

used to query the type of a group insert or instrument insert effect, can be any of the following:

```
$EFFECT_TYPE_FILTER
$EFFECT_TYPE_COMPRESSOR
$EFFECT_TYPE_LIMITER
$EFFECT_TYPE_INVERTER
$EFFECT_TYPE_SURROUND_PANNER
$EFFECT_TYPE_SHAPER {Saturation}
$EFFECT_TYPE_LOFI
$EFFECT_TYPE_STEREO {Stereo Modeller}
$EFFECT_TYPE_DISTORTION
$EFFECT_TYPE_SEND_LEVELS
$EFFECT_TYPE_PHASER
$EFFECT_TYPE_CHORUS
$EFFECT_TYPE_FLANGER
$EFFECT_TYPE_REVERB
$EFFECT_TYPE_DELAY
$EFFECT_TYPE_IRC {Convolution}
$EFFECT_TYPE_GAINER
$EFFECT_TYPE_SKREAMER
$EFFECT_TYPE_ROTATOR
$EFFECT_TYPE_TWANG
$EFFECT_TYPE_CABINET
$EFFECT_TYPE_AET_FILTER

$EFFECT_TYPE_NONE {empty slot}
```

$ENGINE_PAR_SEND_EFFECT_TYPE

used to query the type of a send effect, can be any of the following:

```
$EFFECT_TYPE_PHASER
$EFFECT_TYPE_CHORUS
$EFFECT_TYPE_FLANGER
$EFFECT_TYPE_REVERB
$EFFECT_TYPE_DELAY
$EFFECT_TYPE_IRC {Convolution}
$EFFECT_TYPE_GAINER

$EFFECT_TYPE_NONE {empty slot}
```

$ENGINE_PAR_EFFECT_SUBTYPE

used to query the type of filter/EQ, can be any of the following:

```
$FILTER_TYPE_LP1POLE
$FILTER_TYPE_HP1POLE
$FILTER_TYPE_BP2POLE
$FILTER_TYPE_LP2POLE
$FILTER_TYPE_HP2POLE
$FILTER_TYPE_LP4POLE
$FILTER_TYPE_HP4POLE
$FILTER_TYPE_BP4POLE
$FILTER_TYPE_BR4POLE
$FILTER_TYPE_LP6POLE
$FILTER_TYPE_PHASER
$FILTER_TYPE_VOWELA
$FILTER_TYPE_VOWELB
$FILTER_TYPE_PRO52
$FILTER_TYPE_LADDER
$FILTER_TYPE_VERSATILE
$FILTER_TYPE_EQ1BAND
$FILTER_TYPE_EQ2BAND
$FILTER_TYPE_EQ3BAND
```

$ENGINE_PAR_INTMOD_TYPE

used to query the type of internal modulators, can be any of the following:

```
$INTMOD_TYPE_NONE
$INTMOD_TYPE_LFO
$INTMOD_TYPE_ENVELOPE
$INTMOD_TYPE_STEPMOD
$INTMOD_TYPE_ENV_FOLLOW
$INTMOD_TYPE_GLIDE
```

$ENGINE_PAR_INTMOD_SUBTYPE

used to query the sub type of envelopes and LFOs, can be any of the following:

```
$ENV_TYPE_AHDSR
$ENV_TYPE_FLEX
$ENV_TYPE_DBD

$LFO_TYPE_RECTANGLE
$LFO_TYPE_TRIANGLE
$LFO_TYPE_SAWTOOTH
$LFO_TYPE_RANDO$LFO_TYPE_MULTI
```

Group Start Options Query

Group Start Options Variables

```
$ENGINE_PAR_START_CRITERIA_MODE
$ENGINE_PAR_START_CRITERIA_KEY_MIN
$ENGINE_PAR_START_CRITERIA_KEY_MAX
$ENGINE_PAR_START_CRITERIA_CONTROLLER
$ENGINE_PAR_START_CRITERIA_CC_MIN
$ENGINE_PAR_START_CRITERIA_CC_MAX
$ENGINE_PAR_START_CRITERIA_CYCLE_CLASS
$ENGINE_PAR_START_CRITERIA_ZONE_IDX
$ENGINE_PAR_START_CRITERIA_SLICE_IDX
$ENGINE_PAR_START_CRITERIA_SEQ_ONLY
$ENGINE_PAR_START_CRITERIA_NEXT_CRIT
```

`$ENGINE_PAR_START_CRITERIA_MODE` can return one of the following values:

```
$START_CRITERIA_NONE
$START_CRITERIA_ON_KEY
$START_CRITERIA_ON_CONTROLLER
$START_CRITERIA_CYCLE_ROUND_ROBIN
$START_CRITERIA_CYCLE_RANDOM
$START_CRITERIA_SLICE_TRIGGER
```

`$ENGINE_PAR_START_CRITERIA_NEXT_CRIT` can return one of the following values:

```
$START_CRITERIA_AND_NEXT
$START_CRITERIA_AND_NOT_NEXT
$START_CRITERIA_OR_NEXT
```

Advanced Concepts

Preprocessor & System Scripts

`SET_CONDITION(<condition-symbol>)`

define a symbol to be used as a condition

`RESET_CONDITION(<condition-symbol>)`

delete a definition

```
USE_CODE_IF(<condition-symbol>)
...
END_USE_CODE
```

interpret code when `<condition>` is defined

```
USE_CODE_IF_NOT(<condition-symbol>)
...
END_USE_CODE
```

interpret code when `<condition>` is not defined

`NO_SYS_SCRIPT_GROUP_START`

condition; if defined with `SET_CONDITION()`, the system script which handles all group start options will be bypassed

`NO_SYS_SCRIPT_PEDAL`

condition; if defined with `SET_CONDITION()`, the system script which sustains notes when CC# 64 is received will be bypassed

`NO_SYS_SCRIPT_RLS_TRIG`

condition; if defined with `SET_CONDITION()`, the system script which triggers samples upon the release of a key is bypassed

`reset_rls_trig_counter(<note>)`

resets the release trigger counter (used by the release trigger system script)

`will_never_terminate(<event-id>)`

tells the script engine that this event will never be finished (used by the release trigger system script)

Examples

A preprocessor is used to exclude code elements from interpretation. Here's how it works:

```
USE_CODE_IF(<condition>)
...
END_USE_CODE
```

or

```
USE_CODE_IF_NOT(<condition>)
...
END_USE_CODE
```

`<condition>` refers to a symbolic name which consists of alphanumeric symbols, preceded by a letter. You could write for example:

```
on note
  {do something general}
  $var := 5

  {do some conditional code}
USE_CODE_IF_NOT(dont_do_sequencer)
  while ($count > 0)
        play_note()
  end while
END_USE_CODE
end on
```

What's happening here?
Only if the symbol `dont_do_sequencer` is not defined, the code between `USE_` and `END_USE` will be processed. If the symbol were to be found, the code would not be passed on to the parser; it is as if the code was never written (therefore it does not utilize any CPU power).

You can define symbols with

```
SET_CONDITION(<condition symbol>)
```

and delete the definition with

```
RESET_CONDITION(<condition symbol>)
```

All commands will be interpreted **before** the script is running, i.e. by using `USE_CODE_` the code might get stalled before it is passed to the script engine. That means, `SET_CONDITION` and `RESET_CONDITION` are actually not true commands: they cannot be utilized in `if()...end if` statements; also a `wait()` statement before those commands is useless. Each `SET_CONDITION` and `RESET_CONDITION` will be executed before something else happens.
All defined symbols are passed on to following scripts, i.e. if script 3 contains conditional code, you can turn it on or off in script 1 or 2.

You can use conditional code to bypass system scripts. There are two built-in symbols:

```
NO_SYS_SCRIPT_PEDAL
NO_SYS_SCRIPT_RLS_TRIG
```

If you define one of those symbols with `SET_CONDITION()`, the corresponding part of the system scripts will be bypassed. For clarity reasons, those definitions should always take place in the `init` callback.

```
on init
  {we want to do our own release triggering}
  SET_CONDITION(NO_SYS_SCRIPT_RLS_TRIG)
end on

on release
  {do something custom here}
end on
```

PGS

It is now possible to send and receive values from one script to another, discarding the usual left-to-right order by using the new Program Global Storage (PGS) commands. PGS is a dynamic memory which can be read/written by any script. Here are the commands:

PGS commands

```
pgs_create_key(<key-id>,<size>)
pgs_key_exists(<key-id>)
pgs_set_key_val(<key-id>,<index>,<value>)
pgs_get_key_val(<key-id>,<index>)
```

<key-id> is something similar to a variable name, it can only contain letters and numbers and must not start with a number. It might be a good idea to always write them in capitals to emphasize their unique status.

Here's an example, insert this script into any slot:

```
on init
  pgs_create_key(FIRST_KEY, 1) {defines a key with 1 element}
  pgs_create_key(NEXT_KEY, 128) {defines a key with 128 elements}
  declare ui_button $Just_Do_It
end on

on ui_control($Just_Do_It)

  {writes 70 into the first and only memory location of FIRST_KEY}
  pgs_set_key_val(FIRST_KEY, 0, 70)


  {writes 50 into the first and 60 into the last memory location of NEXT KEY}
  pgs_set_key_val(NEXT_KEY, 0, 50)
  pgs_set_key_val(NEXT_KEY, 127, 60)
end on
```

and insert the following script into any other slot:

```
on init
  declare ui_knob $First (0,100,1)
  declare ui_table %Next[128] (5,2,100)
end on
on pgs_changed

  {checks if FIRST_KEY and NEXT_KEY have been declared}
  if(pgs_key_exists(FIRST_KEY) and _pgs_key_exists(NEXT_KEY))
        $First := pgs_get_key_val(FIRST_KEY,0) {in this case 70}
        %Next[0] := pgs_get_key_val(NEXT_KEY,0) {in this case 50}
        %Next[127] := pgs_get_key_val(NEXT_KEY,127) {in this case 60}
  end if
end on
```

As illustrated above, there is also a new callback type which is executed whenever a set_key command has been executed:

on pgs_changed

callback type, executed whenever any pgs_set_key_val() is executed in any script

It is possible to have as many keys as you want, however each key can only have up to 256 elements.

PGS String Support

The basic handling for PGS strings is the same as for normal PGS keys; there's only one difference: PGS strings keys aren't arrays like the standard PGS keys you already know – they resemble normal string variables.

PGS strings commands

```
pgs_create_str_key(<key-id>)
pgs_str_key_exists(<key-id>)
pgs_set_str_key_val(<key-id>,<stringvalue>)
<stringvalue> := pgs_get_str_key_val(<key-id>)
```

`<key-id>` is something similar to a variable name, it can only contain letters and numbers and must not start with a number. It might be a good idea to always write them in capitals to emphasize their unique status.

Zone and Slice Functions

`find_zone(<zone-name>)`

returns the zone ID for the specified zone name.
Only availabe in the init callback.

`get_sample_length(<zone-ID>)`

returns the length of the specified zone's sample in microseconds

`num_slices_zone(<zone-ID>)`

returns the number of slices of the specified zone

`zone_slice_length(<zone-ID>,<slice-index>)`

returns the length in microseconds of the specified slice with respect to the current tempo

`zone_slice_start(<zone-ID>,<slice-index>)`

returns the absolute start point of the specified slice in microseconds, independent of the current tempo

`zone_slice_idx_loop_start(<zone-ID>,<loop-index>)`

returns the index number of the slice at the loop start

`zone_slice_idx_loop_end(<zone-ID>,<loop-index>)`

returns the index number of the slice at the loop end

`zone_slice_loop_count(<zone-ID>,<loop-index>)`

returns the loop count of the specified loop

`dont_use_machine_mode(<ID-number>)`

play the specified event in sampler mode

User defined Functions

```
function <function-name>
…
end function
```
declares a function

```
call <function-name>
```
calls a previously declares function

Remarks

The function has to be declared before it is called.

Examples

```
on init
   declare $root_note := 60

   declare ui_button $button_1
   set_text ($button_1,"Play C Major")

   declare ui_button $button_2
   set_text ($button_2,"Play Gb Major")

   declare ui_button $button_3
   set_text ($button_3,"Play C7 (b9,#11)")

end on

function func_play_triad
   play_note($root_note,100,0,300000)
   play_note($root_note + 4,100,0,300000)
   play_note($root_note + 7,100,0,300000)
end function

on ui_control ($button_1)
   $root_note := 60
   call func_play_triad
   $button_1 := 0
end on

on ui_control ($button_2)
   $root_note := 66
   call func_play_triad
   $button_2 := 0
end on

on ui_control ($button_3)
   $root_note := 60
   call func_play_triad
   $root_note := 66
   call func_play_triad
   $button_3 := 0
end on
```
Jazz Harmony 101

Working with MIDI files

Please note that this is a VERY rough overview of how to work with MIDI files. We will provide you with additional details and examples in the future.
Handling MIDI files in Kontakt 5 is a very complex matter as there is no internal player functionality – how the incoming MIDI data is handled is up to the scripter.
The first thing you want to do is to load a MIDI file. Please note that you can use only one MIDI file at a time within an NKI. Use the `load_midi_file()` command to load your MIDI file.

Navigation

After loading the MIDI file you are able to navigate through its events by using one of the following commands: `mf_get_first()`, `mf_get_next()` `mf_get_next_at()`, `mf_get_last()`, `mf_get_prev()` and `mf_get_prev_at()`.
By moving to an event you automatically select the event and you are able to edit and change it. But first you might want to know what type of MIDI event you are dealing with. Use the `mf_get_command()` command and compare the returned value to the MIDI constants that you can fine in the Multi Script section. You can receive additional information about the event's settings by using these commands: `mf_get_channel()`, `mf_get_byte_one()`, `mf_get_byte_two()`, `mf_get_pos()` and `mf_get_track_idx()`.

Creating a simple Main Loop

What you want to do next is to create a loop that handles the incoming MIDI data – it might be a good idea to use Kontakt's `on listener` callback for the main loop. Now, jump to the next MIDI event right after the current song position. Then measure the distance between the current song position and the next event in ticks – and wait by using wait ticks. When the current song position has reached the position of the currently selected MIDI event, analyse its type and bytes and use the corresponding commands. For example when your selected event is note_on MIDI event, you should use a `play_note()` command; of your selected event is a note_off message then you should use the `note_off()` command. Of course you are completely free with want to do with the incoming MIDI events – you can transform them as you like.
After that, jump to the next event and wait again. Be sure to check if you have reached the end of the midi file (`mf_get_command()` returns 0).

Some advice

It might make things easier for you if analyse the whole MIDI file right after loading it (please use Kontakt's `on async_complete` callback to be sure that the file has been successfully loaded) and store its data into one or more arrays. This might be especially helpful if you plan to edit the MIDI data.

Editing MIDI data

Kontakt's MIDI features also allow you to edit the loaded MIDI file by changing event types, moving events around etc. You can also store your changes into new MIDI files. Use these commands to edit your events: `mf_set_channel()`, `mf_set_command()`, `mf_set_byte_one()`, `mf_set_byte_two()` and `mf_set_pos()`. To store your edits use the `save_midi_file()` command.

Multi Script

Introduction

The multi script utilizes basically the same KSP syntax as the instrument scripts. Here are the main differences:

- the multi script works on a pure MIDI event basis, i.e. you're working with raw MIDI data
- there are no "on note", "on release" and "on controller" callbacks
- every MIDI event triggers the "**on midi_in**" callback
- there are various built-in variables for the respective MIDI bytes

The new multi script tab is accessed by clicking on the "KSP" button in the multi header.

Just like instrument scripts are saved with the instrument, multi scripts are saved with the multi. GUI-wise everything's identical with the instrument script except for the height, it's limited to 3 grid spaces (just like the instrument scripts in Kontakt 2/3). The scripts are stored in a folder called "multiscripts", which resides next to the already existing "scripts" folder, that is inside the "presets" folder:

/Native Instruments/Kontakt 4/presets/multiscripts

The multi script has only two callback types, the **on midi_in** callback and the various **on ui_control** callbacks. Each MIDI event like Note, Controller, Program Change etc. is triggering the **midi_in** callback.

It is very important to understand the different internal structure of the event processing in the multi script opposed to the instrument script.

On the instrument level, you can retrieve the event IDs of notes only, that is, $EVENT_ID only works in the **on note** and **on release** callback. On the multi level, **any** incoming MIDI event has a unique ID which can be retrieved with $EVENT_ID. This means, $EVENT_ID can be a note event, a controller message, a program change command etc.

This brings us to the usage of change_note(), change_velo() etc. commands. Since $EVENT_ID does not necessarily refer to a note event, this commands will not work in the multi script (there will be a command coming soon which enables you to change the MIDI bytes of events without having to ignore them first).

And most important of all, remember that the multi script is really nothing more than a MIDI processor (where as the instrument script is an event processor). A note event in the instrument script is bound to a voice, whereas MIDI events from the multi script are "translated' into note events on the instrument level. This simply means, that play_note(), change_tune() etc. don't work in the multi script.

Please note that you should be familiar with the basic structure of MIDI messages when working with the multi script.

ignore_midi

ignore_midi
ignores MIDI events

Remarks

Like `ignore_event()`, `ignore_midi` is a very "strong" command. Keep in mind that `ignore_midi` will ignore all incoming MIDI events. If you simply want to change the MIDI channel and/or any of the MIDI bytes, you can also use `set_event_par()`.

Examples

```
on midi_in
  if ($MIDI_COMMAND = $MIDI_COMMAND_NOTE_ON and $MIDI_BYTE_2 > 0)
        ignore_midi
  end if

  if ($MIDI_COMMAND = $MIDI_COMMAND_NOTE_OFF or ...
     ($MIDI_COMMAND = $MIDI_COMMAND_NOTE_ON and $MIDI_BYTE_2 = 0))
        ignore_midi
  end if
end on
```

ignoring note on and note off messages. Note that some keyboards use a note on command with a velocity of 0 to designate a note off command.

See Also

`ignore_event()`

on midi_in

```
on midi_in
```

midi callback, triggered by every incoming MIDI event

Remarks

Like `ignore_event()`, `ignore_midi` is a very "strong" command. Keep in mind that `ignore_midi` will ignore all incoming MIDI events. If you simply want to change the MIDI channel and/or any of the MIDI bytes, you can also use `set_event_par()`.

Examples

```
on midi_in
  if ($MIDI_COMMAND = $MIDI_COMMAND_NOTE_ON and $MIDI_BYTE_2 > 0)
        message ("Note On")
  end if
  if ($MIDI_COMMAND = $MIDI_COMMAND_NOTE_ON and $MIDI_BYTE_2 = 0)
        message ("Note Off")
  end if
  if ($MIDI_COMMAND = $MIDI_COMMAND_NOTE_OFF)
        message ("Note Off")
  end if
  if ($MIDI_COMMAND = $MIDI_COMMAND_CC)
        message ("Controller")
  end if
  if ($MIDI_COMMAND = $MIDI_COMMAND_PITCH_BEND)
        message ("Pitch Bend")
  end if
  if ($MIDI_COMMAND = $MIDI_COMMAND_MONO_AT)
        message ("Channel Pressure")
  end if
  if ($MIDI_COMMAND = $MIDI_COMMAND_POLY_AT)
        message ("Poly Pressure")
  end if
  if ($MIDI_COMMAND = $MIDI_COMMAND_PROGRAM_CHANGE)
        message ("Program Change")
  end if
end on
```

monitoring various MIDI data

See Also

`ignore_midi`

set_midi()

```
set_midi(<channel>,<command>,<byte-1>, <byte-2>)
```

create any type of MIDI event

Remarks

If you simply want to change the MIDI channel and/or any of the MIDI bytes, you can also use `set_event_par()`.

Examples

```
on midi in
   if ($MIDI_COMMAND = $MIDI_COMMAND_NOTE_ON and $MIDI_BYTE_2 > 0)
           set_midi ($MIDI_CHANNEL,$MIDI_COMMAND_NOTE_ON,$MIDI_BYTE_1+4,$MIDI_BYTE_2)
           set midi ($MIDI CHANNEL,$MIDI COMMAND NOTE ON,$MIDI BYTE 1+7,$MIDI BYTE 2)
   end if

   if ($MIDI COMMAND = $MIDI COMMAND NOTE OFF or ...
      ($MIDI_COMMAND = $MIDI_COMMAND_NOTE_ON and $MIDI_BYTE_2 = 0))
           set_midi ($MIDI_CHANNEL,$MIDI_COMMAND_NOTE_ON,$MIDI_BYTE_1+4,0)
           set midi ($MIDI CHANNEL,$MIDI COMMAND NOTE ON,$MIDI BYTE 1+7,0)
   end if
end on
```

a simple harmonizer – notice that you have to supply the correct note off commands as well

See Also

`set_event_par()`
`$EVENT_PAR_MIDI_CHANNEL`
`$EVENT_PAR_MIDI_COMMAND`
`$EVENT_PAR_MIDI_BYTE_1`
`$EVENT_PAR_MIDI_BYTE_2`

Multi Script Variables

$MIDI_CHANNEL

the MIDI channel of the received MIDI event.
Since Kontakt can handle four different MIDI ports, this number can go from 0 - 63 (four ports x 16 MIDI channels)

$MIDI_COMMAND

the command type like Note, CC, Program Change etc. of the received MIDI event.
There are various constants for this variable (see below)

$MIDI_BYTE_1
$MIDI_BYTE_2

the two MIDI bytes of the MIDI message (always in the range 0-127)

$MIDI_COMMAND_NOTE_ON

$MIDI_BYTE_1 = note number
$MIDI_BYTE_2 = velocity
Note: a velocity value of 0 equals a note off command

$MIDI_COMMAND_NOTE_OFF

$MIDI_BYTE_1 = note number
$MIDI_BYTE_2 = release velocity

$MIDI_COMMAND_POLY_AT

$MIDI_BYTE_1 = note number
$MIDI_BYTE_2 = polyphonic key pressure value

$MIDI_COMMAND_CC

$MIDI_BYTE_1 = controller number
$MIDI_BYTE_2 = controller value

$MIDI_COMMAND_PROGRAM_CHANGE

$MIDI_BYTE_1 = program number
$MIDI_BYTE_2 = not used

$MIDI_COMMAND_MONO_AT

$MIDI_BYTE_1 = channel pressure value
$MIDI_BYTE_2 = not used

$MIDI_COMMAND_PITCH_BEND

$MIDI_BYTE_1 = LSB value
$MIDI_BYTE_2 = MSB value

$MIDI_COMMAND_RPN/$MIDI_COMMAND_NRPN

$MIDI_BYTE_1 = rpn/nrpn address
$MIDI_BYTE_2 = rpn/nrpn value

Event Parameter Constants

event parameters to be used with `set_event_par()` and `get_event_par()`

```
$EVENT_PAR_MIDI_CHANNEL
$EVENT_PAR_MIDI_COMMAND
$EVENT_PAR_MIDI_BYTE_1
$EVENT_PAR_MIDI_BYTE_2
```

Resource Container

Introduction

The Resource Container is a very useful tool for library developers. You can see it as a dedicated location to store all of your scripts, graphics, ir_files etc. so that they are no longer hidden within the user folder. Another benefit is that you can create a separate resource container monolith file containing all the scripts, graphics etc. so that you can easily move them around or send them to other team members.

Setup

To create a Resource Container for your NKI, open up its instrument options and click on the <Create> button. After creating a new resource container file, KONTAKT checks if there is already a resource folder structure available. If there isn't just click Yes and let KONTAKT create it for you. You will now find a Resources and a Data folder next to your NKI.
The Resources folder is the place where you can store **all of your needed files**. As you can see KONTAKT has already created several subfolders for you: ir_samples, pictures (for GUI graphics and wallpapers) and scripts. The only thing to do now is to move your files into the right folders and you are ready to go.

Working with the Resource Container

Let's say you're creating a new huge library: after setting up the Resource Container as described above you can tell all of your NKIs that are part of your library to use this special Resource Container. Just open up the NKI's instrument options and use the Browse Function.
As long as the Resources folder exist besides the NKR file (this is the Resource Container monolith), KONTAKT will read all files **directly from this folder structure**.
For loading scripts from the Scripts subfolder, use the "Apply from Container (Info folder)" function within the script editor.
Now let's say you want to send your current working status to another team member. Open up the instrument options, click the Create button and then overwrite your NKR file. Be aware that this will completely **overwrite** your monolith, it won't be matched in any way. Now KONTAKT will do all of the following:

- check the ir_samples subfolder for any .wav, .aif or .aiff files and put them into the monolith.
- check the pictures folder for any .tga or .png files that also have a **.txt file of the same filename** next to them. All of these will be packed into the monolith. Note that **wallpapers also need a .txt file** or they will be ignored.
- check the scripts subfolder for any .txt files which will then be put into the monolith.

After that rename your Resources folder and reopen your NKI. Now as there is no Resources folder present anymore, KONTAKT will **automatically use the NKR monolith file**. If everything is still working as expected you can send your NKIs and the NKR monolith to your team member.
To continue your works just rename the Resources folder back to "Resources".

Remarks

- The Resource Container will be checked in the samples missing dialog.
- When you save your NKI as a monolith file the Resource Container will not be integrated into the monolith – the path to the Resource Container will be saved in absolute path mode.

Version History

Kontakt 5

New Features

- MIDI file support incl. a whole lot of new commands: `load_midi_file()`, `save_midi_file()`, `mf_get_num_tracks()`, `mf_get_first()`, `mf_get_next()`, `mf_get_next_at()`, `mf_get_last()`, `mf_get_prev()`, `mf_get_prev_at()`, `mf_get_channel()`, `mf_get_command()`, `mf_get_byte_one()`, `mf_get_byte_two()`, `mf_get_pos()`, `mf_get_track_idx()`, `mf_set_channel()`, `mf_set_command()`, `mf_set_byte_one()`, `mf_set_byte_two()`, `mf_set_pos()`
- new ui control: `ui_text_edit`
- new ui control: `ui_level_meter`
- new commands for the ui_level_meter: `attach_level_meter()`, `$CONTROL_PAR_BG_COLOR`, `$CONTROL_PAR_OFF_COLOR`, `$CONTROL_PAR_ON_COLOR`, `$CONTROL_PAR_OVERLOAD_COLOR`, `$CONTROL_PAR_PEAK_COLOR`, `$CONTROL_PAR_VERTICAL`
- new ui control: `ui_file_selector`
- new commands for the ui_file_selector: `fs_get_filename()`, `fs_navigate()`, `$CONTROL_PAR_BASEPATH`, `$CONTROL_PAR_COLUMN_WIDTH`, `$CONTROL_PAR_FILEPATH`, `$CONTROL_PAR_FILE_TYPE`
- new commands for dynamic dropdown menus: `get_menu_item_value()`, `get_menu_item_str()`, `get_menu_item_visibility()`, `set_menu_item_value()`, `set_menu_item_str()`, `set_menu_item_visibility()`, `$CONTROL_PAR_SELECTED_ITEM_IDX`, `$CONTROL_PAR_NUM_ITEMS`
- new callback type: `on async_complete`
- a new internal bus system with a new internal constant: `$NI_BUS_OFFSET`
- new internal effects with new engine_par constants: $ENGINE_PAR_TP_GAIN, $ENGINE_PAR_TP_WARMTH, $ENGINE_PAR_TP_HF_ROLLOFF, $ENGINE_PAR_TR_INPUT, $ENGINE_PAR_TR_ATTACK, $ENGINE_PAR_TR_SUSTAIN, $ENGINE_PAR_SEQ_LF_GAIN, $ENGINE_PAR_SEQ_LF_FREQ, $ENGINE_PAR_SEQ_LF_BELL, $ENGINE_PAR_SEQ_LMF_GAIN, $ENGINE_PAR_SEQ_LMF_FREQ, $ENGINE_PAR_SEQ_LMF_Q, $ENGINE_PAR_SEQ_HMF_GAIN, $ENGINE_PAR_SEQ_HMF_FREQ, $ENGINE_PAR_SEQ_HMF_Q, $ENGINE_PAR_SEQ_HF_GAIN, $ENGINE_PAR_SEQ_HF_FREQ, $ENGINE_PAR_SEQ_HF_BELL, $ENGINE_PAR_SCOMP_THRESHOLD, $ENGINE_PAR_SCOMP_RATIO, $ENGINE_PAR_SCOMP_ATTACK, $ENGINE_PAR_SCOMP_RELEASE, $ENGINE_PAR_SCOMP_MAKEUP, $ENGINE_PAR_SCOMP_MIX
- new internal variables for the new callback: `$NI_ASYNC_ID`, `$NI_ASYNC_EXIT_STATUS`, `$NI_CB_TYPE_ASYNC_OUT`
- new wait commands: `wait_ticks()`, `stop_wait()`
- support for string arrays added to save / load array
- PGS support for strings: `pgs_create_str_key()`, `pgs_str_key_exists()`, `pgs_set_str_key_val()`, `pgs_get_str_key_val()`
- the maximum height of `set_ui_height_px()` is now 540 pixels

Kontakt 4.2

New Features

- the Resource Container, a helpful tool when creating new libraries
- new ID to set wallpapers via script: `$INST_WALLPAPER_ID`
- a new key color: `$KEY_COLOR_BLACK`
- new callback type: `on listener`
- new commands for this callback: `set_listener(), change_listener_par()`
- new commands for storing arrays: `save_array(), load_array()`
- a new command to check the purge status of a group: `get_purge_state()`
- new built-in variable: `$NI_SONG_POSITION`
- new control parameter: `$CONTROL_PAR_ALLOW_AUTOMATION`

Improved Features

- The script editor is now much more efficient, especially with large scripts.
- New ui control limit: 256 (per control and script).
- Event parameters can now be used without influencing the system scripts.

Kontakt 4.1.2

New Features

- new UI control: `UI waveform`
- new commands for this UI control: `set_ui_wf_property(), get_ui_wf_property(),attach_zone()`
- new variables & constants to be used with these commands: `$UI_WAVEFORM_USE_SLICES, $UI_WAVEFORM_USE_TABLE, $UI_WAVEFORM_TABLE_IS_BIPOLAR, $UI_WAVEFORM_USE_MIDI_DRAG, $UI_WF_PROP_PLAY_CURSOR, $UI_WF_PROP_FLAGS, $UI_WF_PROP_TABLE_VAL, $UI_WF_PROP_TABLE_IDX_HIGHLIGHT, $UI_WF_PROP_MIDI_DRAG_START_NOTE`
- new event parameter: `$EVENT_PAR_PLAY_POS`

Kontakt 4.1.1

Improved Features

- The built-in variables `$SIGNATURE_NUM` and `$SIGNATURE_DENOM` don't reset to 4/4 if the host's transport is stopped

Kontakt 4.1

New Features

- implementation of user defined functions: `function`
- new control parameter variable: `$CONTROL_PAR_AUTOMATION_NAME`
- new command: `delete_event_mark()`

- support for polyphonic aftertouch:
  `on poly_at…end on, %POLY_AT[], $POLY_AT_NUM`
- new command: `get_event_ids()`
- new control parameter variables:
  `$CONTROL_PAR_KEY_SHIFT, $CONTROL_PAR_KEY_ALT, $CONTROL_PAR_KEY_CONTROL`

Improved Features

- The built-in variable `$MIDI_CHANNEL` is now also supported in the instrument script.
- The sample offset parameter in `play_note()` now also works in DFD mode, according to the S.Mod value set for the respectives zone in the wave editor

Manual Corrections

- correct Modulation Engine Parameter Variables

Kontakt 4.0.2

New Features

- new engine parameter to set the group output channel: `$ENGINE_PAR_OUTPUT_CHANNEL`
- new built-in variable: `$NUM_OUTPUT_CHANNELS`
- new function: `output_channel_name()`
- new built-in variable: `$CURRENT_SCRIPT_SLOT`
- new built-in variable: `$EVENT_PAR_SOURCE`

Improved Features

- The `load_ir_sample()` command now also accepts single file names for loading IR samples into KONTAKT's convolution effect, i.e. without a path designation. In this case the sample is expected to reside in the folder called "ir_samples" inside the user folder.

Kontakt 4

New Features

- Multiscript
- New id-based User Interface Controls system:
  `set_control_par()`, `get_control_par()` and `get_ui_id()`
- Pixel exact positioning and resizing of UI controls
- Skinning of UI controls
- New UI controls: switch and slider
- Assign colors to Kontakt's keyboard by using `set_key_color()`
- new timing variable: `$KSP_TIMER` (in microseconds)
- new path variable: `$GET_FOLDER_FACTORY_DIR`
- new hide constants: `$HIDE_PART_NOTHING` & `$HIDE_WHOLE_CONTROL`
- link scripts to text files

Improved Features

- New array size limit: 32768
- Retrieve and set event parameters for tuning, volume and pan of an event (`$EVENT_PAR_TUNE`, `$EVENT_PAR_VOL` and `$EVENT_PAR_PAN`)
- larger performance view size, `set_ui_height()`, `set_script_title()`
- beginning underscores from Kontakt 2/3 commands like `_set_engine_par()` can be omitted, i.e. you can write `set_engine_par()` instead

Kontakt 3.5

New Features

- Retrieve the status of a particular event: `event_status()`
- Hide specific parts of UI controls: `hide_part()`
  `%GROUPS_SELECTED`

Improved Features

- Support for channel aftertouch: `$VCC_MONO_AT`
- New array size limit: 2048

Kontakt 3

New Features

- Offset for wallpaper graphic: `_set_skin_offset()`
- Program Global Storage (PGS) for inter-script communication
  `_pgs_create_key()`
  `_pgs_key_exists()`
  `_pgs_set_key_val()`
  `_pgs_get_key_val()`
- New callback type: `on _pgs_changed`
- Adressing modulators by name:
  `find_mod()`
  `find_target()`
- Change the number of displayed steps in a column: `set_table_steps_shown()`
- Info tags for UI controls: `set_control_help()`

Improved Features

- All five performance views can now be displayed together

Kontakt 2.2

New Features

- New callback type: `on ui_update`
- New built-in variables for group based scripting
  `$REF_GROUP_IDX`
  `%GROUPS_SELECTED`
- Ability to create custom group start options:
  `NO_SYS_SCRIPT_GROUP_START`
  (+ various Group Start Options Variables)
- Retrieving the release trigger state of a group: `$ENGINE_PAR_RELEASE_TRIGGER`
- Default values for knobs: `set_knob_defval()`

Kontakt 2.1.1

New Features

- Assign unit marks to knobs: `set_knob_unit()`
- Assign text strings to knobs: `set_knob_label()`
- Retrieve the knob display: `_get_engine_par_disp()`

Kontakt 2.1

New Features

- string arrays (! prefix) and string variables (@ prefix)
- engine parameter: `_set_engine_par()`
- loading IR samples: `_load_ir_sample()`
- Performance View: `make_perfview`
- rpn/nrpn implementation:
  `on rpn` & `on nrpn`
  `$RPN_ADDRESS`
  `$RPN_VALUE`
  `msb()` & `lsb()`
  `set_rpn()` & `set_nrpn()`
- event parameters: `set_event_par()`
- New built-in variables:
  `$NUM_GROUPS`
  `$NUM_ZONES`
  `$VCC_PITCH_BEND`
  `$PLAYED_VOICES_TOTAL`
  `$PLAYED_VOICES_INST`

Improved Features

- possible to name UI controls with `set_text()`
- moving and hiding UI controls
- MIDI CCs generated by `set_controller()` can now also be used for automation (as well as modulation).

Kontakt 2

Initial release.

Index

M

N

O

P

R

S

T

U

W