KSP Reference Manual

Reference Manual written by: Nicki Marinic, Adam Hanley
KONTAKT Version: 5.4.2
Last changed: October 6, 2014

Table of Contents

Callbacks

General Information

- A callback is a section within a script that is being "called" (i.e. executed) at certain times.
- All callbacks start with `on <callback-name>` and end with `end on`
- Callbacks can be stopped by using the command `exit`.
- Each callback has a unique ID number which can be retrieved with `$NI_CALLBACK_ID`
- You can query which callback triggered a function with `$NI_CALLBACK_TYPE` and the corresponding built-in constants.

Examples

```
function show_callback_type
   if ($NI_CALLBACK_TYPE = $NI_CB_TYPE_NOTE)
      message("Function was called from note callback!")
   end if
   if ($NI_CALLBACK_TYPE = $NI_CB_TYPE_CONTROLLER)
      message("Function was called from controller callback!")
   end if
end function

on note
   call show_callback_type
end on

on controller
   call show_callback_type
end on
```

query the callback type in a function

See Also

`exit`
`$NI_CALLBACK_ID`
`$NI_CALLBACK_TYPE`

on async_complete

```
on async_complete
```

async complete callback, triggered after the execution of any load/save command

Remarks

To resolve synchronization issues, the commands mentioned above 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)
   $load_midi_file_id := load_midi_file(<midifile-path>)
   while ($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
```

example that pauses the ui_control callback until the MIDI file is loaded

See Also

`$NI_ASYNC_EXIT_STATUS`
`$NI_ASYNC_ID`
`Load/Save Commands`

on controller

on controller
MIDI 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
```

initialization callback, executed when the script was successfully analyzed

Remarks

The init callback will be executed when:

- clicking the "Apply" button in the script editor
- loading a script preset or an instrument
- restarting KONTAKT's audio engine by clicking the restart button in the Monitor/Engine tab or the restart button in KONTAKT's header
- loading a snapshot with `set_snapshot_type()` set to 0

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()

```
on init
   declare ui_button $Sync
   move_control ($Sync,1,1)
   make_persistent ($Sync)

   declare ui_menu $time
   add_menu_item ($time,"16th",0)
   add_menu_item ($time,"8th",1)
   move_control ($time,0,0)
   make_persistent ($time)
end on

function show_menu
   if ($Sync = 1)
      move_control ($time,2,1)
   else
      move_control ($time,0,0)
   end if
end function

on persistence_changed
   call show_menu
end on

on ui_control ($Sync)
   call show_menu
end on
```

the same script functionality, now with persistence_changed callback

See Also

```
make_persistent()
read_persistent_var()
on persistence_changed
```

on listener

```
on listener
```

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

Remarks

The listener callback is executed at time intervals defined with the `set_listener()` command. It can also react to the host's transport start and stop command. This makes it the ideal callback for anything tempo synced like sequencers, arpeggiators, midi file player etc.

- In some situations (like tempo changes within the host) ticks can be left out.

Examples

```
on init
   declare ui_knob $Test (0,99,1)
   declare $direction
   declare $tick_counter
   set_listener($NI_SIGNAL_TIMER_MS,10000)
end on

on listener
   if ($NI_SIGNAL_TYPE = $NI_SIGNAL_TIMER_MS)
      if ($direction = 0)
         inc($tick_counter)
      else
         dec($tick_counter)
      end if

      $Test := $tick_counter

      if ($tick_counter = 99)
         $direction := 1
      end if
      if ($tick_counter = 0)
         $direction := 0
      end if
   end if
end on
```

not useful as such, but nice to look at

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 Nr: " & $EVENT_NOTE & " - Velocity: " & $EVENT_VELOCITY)
end on
```

query note data

See Also

on release
ignore_event()
set_event_par()
get_event_par()
$EVENT_NOTE
$EVENT_VELOCITY
$EVENT_ID

on persistence_changed

```
on persistence_changed
```

executed after the init callback or whenever a snapshot has been loaded

Remarks

The on persistence_changed callback is called whenever the persistent variables change in an instrument, i.e. it is always executed after the init callback has been called and/or upon loading a snapshot.

Examples

```
on init

   set_snapshot_type(1) {init callback not executed upon snapshot loading}
   reset_ksp_timer

   declare $init_flag {1 if init callback has been executed, 0 otherwise}
   $init_flag := 1

   declare ui_label $label (2,2)
   set_text($label,"init callback " & $KSP_TIMER)
end on

function add_text
   add_text_line($label,"persistence_changed callback " & $KSP_TIMER)
end function

on persistence_changed
   if ($init_flag = 1) {instrument has been loaded}
      call add_text
   else {snapshot has been loaded}
      set_text($label,"Snapshot loaded")
   end if

   $init_flag := 0
end on
```

query if a snapshot or if an instrument has been loaded – also demonstrates the ability to call functions upon initialization, i.e. the persistence callback acts as an extension to the init callback

See Also

```
on init
read_persistent_var()
set_snapshot_type()
```

on pgs_changed

```
on pgs_changed
```

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_create_key()`
`pgs_set_key_val()`
`pgs_get_key_val()`

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 polyphonic $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

This command is triggered with every GUI change in KONTAKT, so use it with caution.

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

General Information

- All user defined variables must be declared in the `on init` callback.
- Variable names may contain only numbers, characters and the underscore (_).
- Variable names are case-sensitive.
- Please do not create variables with the prefixes below, as as these prefixes are used for internal variables and constants

```
$NI_
$CONTROL_PAR
$EVENT_PAR_
$ENGINE_PAR_
```

$ (variable)

```
declare $<variable>
```

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

Examples

```
on init
   declare $test
   $test := -1
end on
```

creating a variable

```
on init
   declare $test := -1
end on
```

creating a variable, same as above but shorter

See Also

on init
make_persistent()
read_persistent_var()

const $ (constant)

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

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

Remarks

- As the name implies, the value of constant variables can only be read, not changed.
- It is quite common to capitalize the names of constants.

Examples

```
on init
   declare const $NUM_OF_PRESETS := 10
   declare const $NUM_OF_PARAMETERS := 5

   declare %preset_data[$NUM_OF_PRESETS * $NUM_OF_PARAMETERS]

end on
```

creating constants, useful when creating preset arrays

See Also

`on init`

polyphonic $ (polyphonic variable)

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

declare a user-defined polyphonic variable to store a single integer value per note event

Remarks

- A polyphonic variable acts as a unique variable for each executed note event, avoiding conflicts in callbacks that are executed in parallel for example when using `wait()`.
- A polyphonic variable retains its value in the release callback of the corresponding note.
- Polyphonic variables need much more memory than normal variables.
- Polyphonic variables should only be used in note and release callbacks.

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

```
on init
   declare $counter
   declare polyphonic $polyphonic_counter
end on

on note
   message($polyphonic_counter & "  " & $counter)
   inc($counter)
   inc($polyphonic_counter)
end on
```

Since a polyphonic variable is always unique per callback, $polyphonic_counter will always be 0 in the displayed message

% (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.
- The number of elements must be defined with a constant value, a standard variable cannot be used.
- It is possible to initialize an array with one value (see the second example below).

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

```
on init
   declare %presets[10*8] := (4)
end on
```

quick way of initializing the same array with the value 4.

See Also

Array and Group Commands
make_persistent()

@ (string variable)

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

declare a user-defined string variable to store text

Remarks

- You cannot declare and define a string variable in the same line of code as you can with an integer variable.
- It is possible to make string variables persistent.

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)
ui_text_edit
make_persistent()
```

! (string array)

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

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

Remarks

- Just like with string variables, the contents of a string array cannot be defined on the same line as the declaration.
- String arrays cannot be made persistent.

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)

make_instr_persistent()

```
make_instr_persistent(<variable>)
```

retain the value of a variable only with the instrument

Remarks

`make_instr_persistent()` is similar to `make_persistent()`, however the value of a variable is only saved with the instrument, not with snapshots. It can be used to e.g. prevent UI elements from being changed when loading snapshots.

Examples

```
   on init

   set_snapshot_type(1) {init callback not executed upon snapshot loading}

   declare ui_knob $knob_1 (0,2,1)
   set_text($knob_1,"Pers.")
   make_persistent($knob_1)

   declare ui_knob $knob_2 (0,2,1)
   set_text($knob_2,"Inst Pers.")
   make_instr_persistent ($knob_2)

   declare ui_knob $knob_3 (0,2,1)
   set_text($knob_3,"Not Pers.")

end on
```

the second knob will not be changed when loading snapshots

See Also

`read_persistent_var()`
`make_persistent()`
`set_snapshot_type()`

make_persistent()

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

retain the value of a variable whith the instrument and snapshot

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()`.
- You can also use the `on persistence` callback for retrieving the values of persistent variables
- When replacing script code by copy and replacing the text, the values of persistent variables are also retained.
- Sometimes, when working on more complex scripts, you'll want to "flush" the variables by resetting the script. You can do this by 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()
on persistence_changed
make_instr_persistence()
```

read_persistent_var()

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

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

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. You can do this by applying an empty script in the respective slot.
- You can also use the `on persistence` callback for retrieving the values of persistent variables

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()
on persistence_changed
```

User Interface Controls

ui_button

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

create a user interface button

Remarks

- a button (i.e. its callback) is triggered when releasing the mouse (aka mouse-up)
- a button 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_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 divided 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

ui_file_selector

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

create a file selector

Remarks

Only one file selector can be applied per script slot.

Examples

(see next page)

```
on init
   set_ui_height(5)

   declare @basepath
   {set browser path here, for example
   @basepath := "/Users/username/Desktop/MIDI Files/"}

   declare @file_name
   declare @file_path

   declare ui_file_selector $file_browser
   declare $browser_id
   $browser_id := get_ui_id($file_browser)

   set_control_par_str($browser_id,$CONTROL_PAR_BASEPATH,@basepath)
   set_control_par($browser_id,$CONTROL_PAR_FILE_TYPE,$NI_FILE_TYPE_MIDI)
   set_control_par($browser_id,$CONTROL_PAR_COLUMN_WIDTH,180)
   set_control_par($browser_id,$CONTROL_PAR_HEIGHT,170)
   set_control_par($browser_id,$CONTROL_PAR_WIDTH,550)
   move_control_px($file_browser,66,2)

   declare ui_button $prev
   declare ui_button $next
   move_control($prev,5,1)
   move_control($next,6,1)

   declare $load_mf_id
   $load_mf_id := -1

end on
on async_complete
   if ($NI_ASYNC_ID = $load_mf_id)
      $load_mf_id := -1
      if ($NI_ASYNC_EXIT_STATUS = 0)
         message("MIDI file not found!")
      else
         message("Loaded MIDI File: " & @file_name)
      end if
   end if
end on
on ui_control ($file_browser)
   @file_name := fs_get_filename($browser_id,0)
   @file_path := fs_get_filename($browser_id,2)
   $load_mf_id := load_midi_file(@file_path)
end on
on ui_control ($prev)
   fs_navigate($browser_id,0)
   @file_name := fs_get_filename($browser_id,0)
   @file_path := fs_get_filename($browser_id,2)
   $load_mf_id := load_midi_file(@file_path)
   $prev := 0
end on
on ui_control ($next)
   fs_navigate($browser_id,1)
   @file_name := fs_get_filename($browser_id,0)
   @file_path := fs_get_filename($browser_id,2)
   $load_mf_id := load_midi_file(@file_path)
   $next := 0
end on
```

loading MIDI files via ui file selector

ui_label

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

create a user interface text label

<width>	the width of the label in grid units
<height>	the height of the label in grid units

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

- The level meter can only be attached to the output levels of buses or the instrument master.

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 channel 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 drop-down 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()
get_menu_item_str()
get_menu_item_value()
get_menu_item_visibility()
set_menu_item_str()
set_menu_item_value()
set_menu_item_visibility()

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
set_control_par()
$CONTROL_PAR_MOUSE_BEHAVIOUR

ui_switch

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

create a user interface switch

Remarks

- a switch (i.e. its callback) is triggered when clicking the mouse (aka mouse-down)
- a switch can be automated

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 grid units
<height>	the height of the table in grid units
<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

Examples

```
on init

   declare ui_text_edit @label_name
   make_persistent(@label_name)

   set_control_par_str(get_ui_id(@label_name),$CONTROL_PAR_TEXT,"empty")
   set_control_par(get_ui_id(@label_name),$CONTROL_PAR_FONT_TYPE,25)
   set_control_par(get_ui_id(@label_name),$CONTROL_PAR_POS_X,73)
   set_control_par(get_ui_id(@label_name),$CONTROL_PAR_POS_Y,2)

   declare ui_label $pattern_lbl(1,1)
   set_text($pattern_lbl,"")
   move_control_px($pattern_lbl,66,2)

end on

on ui_control (@label_name)
   message(@label_name & " it is!")
end on
```

a text edit field on top of a label

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 divided by <display-ratio> for display purposes You can also use $VALUE_EDIT_MODE_NOTE_NAMES to display note names instead of numbers.

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 grid units
<height>	the height of the waveform in grid units

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

Control Statements, Arithmetic Commands & Operators

if…else…end if

if…else…end if
A conditional if statement

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
```

display different messages depending on the controller number.

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
wait()

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()

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()

inc()

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

increment an expression by 1

Examples

```
on init
   declare $count
   declare ui_table %table[100] (6,2,100)

   $count := 0
   while ($count < num_elements(%table))
      %table[$count] := $count
      inc($count)
   end while

end on
```

populating a table by using a while loop

See Also

abs()
dec()
while()

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

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

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

Arithmetic Operators

Arithmetic operators	
x := y	assignment (the value of y is assigned to x)
x + y	addition
x - y	subtraction
x * y	multiplication
x / y	division
x mod y	modulo
-x	negative value

Remarks

- The KONTAKT script language only uses integer values, so be careful when using arithmetic operators. When dividing integers, the KSP engine truncates the remainder, rather than rounding the value. So, for example, the equation 11 / 4 will result in a value of 2, even though the actual result of 2.75 is closer to 3.

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()

Boolean Operators

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

Remarks

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

General Commands

exit

```
exit
```

immediately stops a callback or exits a function

Remarks

- `exit` is a very strong command. Be 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()`
`stop_wait()`

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()
on controller
```

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("")` at the start of 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.
- Messages defined in the init callback will only be displayed if the user manually applies the script by clicking on the APPLY button. These messages will not be displayed when an instrument loads and initializes the script automatically.

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()

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 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()`

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>	sample offset in microseconds
<duration>	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 Sample Mod (S.Mod) value of the respective zones. Sample offset value greater than the zone's S.Mod setting will be ignored and no sample offset will be applied.
- You can retrieve the event ID of the played note in a variable 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()`

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_rpn()/set_nrpn

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

Remarks

- 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_snapshot_type()

set_snapshot_type(<type>)	
configures the KSP processor behavior of all five slots when a snapshot is recalled	
<type>	the available types are: 0: the init callback will always be executed upon snapshot change, afterwards the on persistence_changed callback will be executed (default behavior) 1: the init callback will not be executed upon loading a snapshot, only the on persistence_callback will be executed

Remarks

- This command acts globally, i.e. it can applied in any script slot.
- In snapshot type 1, the value of a non persistent and instrument persistence variable is preserved.
- Loading a snapshot always resets KONTAKT's audio engine, i.e. audio is stopped and all active events are deleted.

Examples

```
on init
    set_snapshot_type(1)

    declare ui_knob $knob_1 (0,127,1)
    set_text($knob_1,"Knob")
    make_persistent($knob_1)

    declare ui_button $gui_btn
    set_text($gui_btn,"Page 1")
end on
function show_gui
    if ($gui_btn = 1)
        set_control_par(get_ui_id($knob_1),$CONTROL_PAR_HIDE,...
        $HIDE_PART_NOTHING)
    else
        set_control_par(get_ui_id($knob_1),$CONTROL_PAR_HIDE,$HIDE_WHOLE_CONTROL)
    end if
end function
on persistence_changed
    call show_gui
end on
on ui_control ($gui_btn)
    call show_gui
end on
```

retaining the GUI upon loading snapshots

See Also

```
on init
on persistence_changed
```

Event Commands

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_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 `$EVENT_NOTE`
- it is not possible to address events via event groups like `$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 `$EVENT_VELOCITY`
- it is not possible to address events via event groups like `$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()

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 assigned to the event.

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()`

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> <parameter>	the ID number of the event 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 $EVENT_PAR_ZONE_ID (use with caution, see below)

Remarks

A note event always "arries 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()`.

Examples

(see next page)

```
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)

```
on note
   wait(1)
   message(get_event_par($EVENT_ID,$EVENT_PAR_ZONE_ID))
end on
```

Note that in the above example, an event itself does not carry a zone ID (only a voice has zone IDs), therefore you need to insert `wait(1)` in order to retrieve the zone ID.

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 retrieving 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 disallowed.

Examples

```
on init
   declare $count
   declare ui_label $label (2,4)
   set_text ($label,"")
end on

on note
   set_text($label,"")
   $count := 0
   while($count < $NUM_GROUPS)

      if (get_event_par_arr($EVENT_ID,$EVENT_PAR_ALLOW_GROUP,$count) = 1)
         add_text_line($label,"Group ID " & $count & " allowed")
      else
         add_text_line($label,"Group ID " & $count & " disallowed")
      end if

      inc($count)
   end while
end on
```

a simple group monitor

See Also

set_event_par_arr()
get_event_par()
$EVENT_PAR_ALLOW_GROUP
%GROUPS_AFFECTED

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()
```

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 assigned to the event. You can also assign 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> <parameter>	the ID number of the event 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` can be used
`<value>`	If set to `1`, the group set with <groupindex> will be allowed for the event. If set to `0`, the group set with <groupindex> 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

`allow_group()`
`disallow_group()`
`get_event_par_arr()`
`set_event_par()`
`$EVENT_PAR_ALLOW_GROUP`

Array Commands

array_equal()

```
array_equal(<array-variable>,<array-variable>)
```

checks 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>)
```

returns 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>)
```

sorts 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()

Group Commands

allow_group()

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

allows the specified group, i.e. makes it available for playback

Remarks

- 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()
set_event_par_arr()

disallow_group()

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

disallows the specified group, i.e. makes it unavailable for playback

Remarks

- 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()
set_event_par_arr()

find_group()

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

returns the group index for the specified group name

Remarks

If no group with the specified name is found, this command will return the value zero. This can cause problems as this is the group index of the first group, so be careful when using this command.

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>)	
returns the purge state of the specified group: 0: the group is purged (0) 1: the group is not purged, i.e. the samples are loaded	
<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

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

   $count := 0
   while ($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()

purge_group()

purge_group(<group-index>,<mode>)	
purges (i.e. unloads from RAM) 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

- 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 and persistence_changed callback
- It is recommended to not use the purge_group() command in the callback of an automatable control.

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

Time-Related Commands

change_listener_par()

change_listener_par(<signal-type>,<parameter>)	
changes the parameters of the `on listener` callback. Can be used in every callback.	

`<signal-type>`	The signal to be changed, can be either:
	`$NI_SIGNAL_TIMER_MS`
	`$NI_SIGNAL_TIMER_BEAT`

`<parameter>`	dependent on the specified signal type:
	
	`$NI_SIGNAL_TIMER_MS`
	time interval in microseconds
	
	`$NI_SIGNAL_TIMER_BEAT`
	time interval in fractions of a beat/quarter note

Examples

```
on init

   declare ui_value_edit $Tempo (20,300,1)
   $Tempo := 120

   declare ui_switch $Play

   set_listener($NI_SIGNAL_TIMER_MS,60000000 / $Tempo)

end on

on listener
   if ($NI_SIGNAL_TYPE = $NI_SIGNAL_TIMER_MS and $Play = 1)
      play_note(60,127,0,$DURATION_EIGHTH)
   end if
end on

on ui_control($Tempo)
   change_listener_par($NI_SIGNAL_TIMER_MS,60000000 / $Tempo)
end on
```

a very basic metronome

See Also

`on listener`
`set_listener()`
`$NI_SIGNAL_TYPE`

ms_to_ticks()

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

converts a microseconds value into a tempo dependent ticks value

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

set_listener()

```
set_listener(<signal-type>,<parameter>)
```

Sets the signals on which the listener callback should react to. Can only be used in the init callback.

`<signal-type>`	the event on which the 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>`	user defined parameter, dependant on the specified signal type:
	
	`$NI_SIGNAL_TIMER_MS`
	time interval in microseconds
	
	`$NI_SIGNAL_TIMER_BEAT`
	time interval in fractions of a beat/quarter note
	
	`$NI_SIGNAL_TRANSP_START`
	set to 1 if the listener callback should react to the host's transport start command
	
	`$NI_SIGNAL_TRANSP_STOP`
	set to 1 if the listener callback should react to the host's transport stop command

Remarks

When using `$NI_SIGNAL_TIMER_BEAT`, the maxium resolution is 24 ticks per beat/quarter note.

Examples

```
on init
   set_listener($NI_SIGNAL_TIMER_BEAT,1)
end on
on listener
   if ($NI_SIGNAL_TYPE = $NI_SIGNAL_TIMER_BEAT)
      message($ENGINE_UPTIME)
   end if
end on
```

triggering the listener callback every beat – also gets triggered even when transport is stopped

See Also

`change_listener_par()`
`$NI_SIGNAL_TYPE`

stop_wait()

stop_wait(<callback-ID>,<parameter>)	
stops wait commands in the specified callback	
<callback-ID> <parameter>	the callback's ID number in which the wait commands will be stopped 0: stops only the current wait 1: stops the current wait and ignores all following wait commands in this callback.

Remarks

- Be careful with while loops when stopping all wait commands in a callback.

Examples

```
on init
   declare ui_button $Play
   declare $id
end on
on ui_control ($Play)
   if ($Play = 1)
      $id := $NI_CALLBACK_ID
      play_note(60,127,0,$DURATION_QUARTER)

      wait($DURATION_QUARTER)
      if ($Play = 1)
         play_note(64,127,0,$DURATION_QUARTER)
      end if

      wait($DURATION_QUARTER)
      if ($Play = 1)
         play_note(67,127,0,$DURATION_QUARTER)
      end if
   else
      stop_wait($id,1)
      fade_out($ALL_EVENTS,10000,1)
   end if
end on
```

the Play button triggers a simple triad arpeggio – without the stop_wait() command, parallel callbacks could occur when pressing the Play button quickly after each other resulting in multiple arpeggios

See Also

wait()
wait_ticks()
Callback Type Variables and Constants (Built-in variables/Specific)

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.
- Since the KSP timer is based on the CPU clock, the main reason to use it is for debugging and optimization. It is a great tool to measure the efficiency of certain script passages. However, it should not be used for 'musical' timing, as it remains at a real-time constant rate, even if KONTAKT is being used in an offline bounce.

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`

ticks_to_ms()

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

converts a tempo dependent ticks value into a microseconds value

Remarks

- Since the returned value is in microseconds, the command will reach its limit after about 30 minutes due to its 32 bit nature.

Examples

```
on init
   declare ui_label $label (2,1)
   declare $msec
   declare $sec
   declare $min
   set_listener($NI_SIGNAL_TIMER_MS,1000)
end on

on listener
   if ($NI_SIGNAL_TYPE = $NI_SIGNAL_TIMER_MS)
      $msec := ticks_to_ms($NI_SONG_POSITION)/1000
      $sec := $msec/1000
      $min := $sec/60
      set_text($label,$min & ":" & $sec mod 60 & "." & $msec mod 1000)
   end if
end on
```

displaying the song position in real time

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

Same as `wait()`, but with ticks as the wait time parameter.

See Also

`stop_wait()`
`wait()`

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 create menu entries only in the init callback but you can change their text and value afterwards by using `set_menu_item_str()` and `set_menu_item_value()`. You can add as many menu entries as you want and then show or hide them dynamically by using `set_menu_item_visibility()`.
- Using the `$CONTROL_PAR_VALUE` constant in the `get_control_par()` command will return the menu index and not the value, if you want to get the menu value, use the `get_menu_item_value()` command.

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

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(<ui-ID>,<group>,<slot>,<channel>,<bus>)	
attach a level meter to a certain position within the instrument to read volume data	
<ui-ID>	the ID number of the level meter
<group>	the index of the group you wish to access. Should be set to -1 if not using the group level
<slot>	the index of the fx slot you wish to access. Should be set to -1 if you do not wish to access an fx slot.
<channel>	select either the left (0) or right (1) channel
<bus>	the index of the instrument bus you wish to access. Should be set to -1 if you are not accessing the bus level.

Remarks

- Currently, the level meters can only be attached to the output level of the instrument buses and the instrument master. Consequently, the group index and slot index should always be set to -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 number of the zone that you want to attach to the waveform display
<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

Remarks

- Use the bitwise `.or.` to combine flags.
- The `$UI_WAVEFORM_USE_TABLE` and `$UI_WAVEFORM_USE_MIDI_DRAG` flags will only work if `$UI_WAVEFORM_USE_SLICES` is already set.

Examples

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

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

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}

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
<return-parameter>	0: returns the filename without extension 1: returns the filename with extension 2: returns the whole path

Examples

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
<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

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, $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_par_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 (not value) of the menu item

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 item

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_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()`
`set_ui_height_px()`

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 grid units
<y-position>	the vertical position of the control (0 to 16) in grid units

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()
$CONTROL_PAR_HIDE

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, for example $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_par_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_knob_defval()

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

assign a default value to a knob to which the knob is reset when Cmd-clicking (mac) or Ctrl-clicking (PC) 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 index of the menu item
<string>	the text you wish to set for the selected menu item

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 index of the menu item
<value>	the value you want to give the menu item

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>,<visibility>)
```

sets the visibility of a menu entry.

<menu-id>	the ID of the menu that you want to modify
<index>	the index of the menu item
<visibility>	set to either 0 (invisible) or 1 (visible)

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().
If you set the currently selected menu item to invisible, the item will remain visible until it is no longer selected.

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 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_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 script title

Remarks

- This command overrides any manually set script titles.

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 picture file by the specified number of pixels

Remarks

If a background tga/png graphic file has been selected in the instrument options and it 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 while only using one picture file.

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 5/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 display 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()
$CONTROL_PAR_TEXT
set_control_par_str()

set_ui_height()

set_ui_height(<height>)	
set the height of a script performance view in grid units	
<height>	the height of script in grid units (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 performance view in pixels	
<height>	the height of script in pixels (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

Keyboard Commands

get_key_color()

```
get_key_color(<note-nr>)
```

returns the color constant of the specified note number

Examples

```
on init
   message("")
   declare $count
   while ($count < 128)
      set_key_color($count,$KEY_COLOR_INACTIVE)
      inc($count)
   end while

   declare $random_key
   $random_key := random(60,71)

   set_key_color($random_key,$KEY_COLOR_RED)

end on

on note
   if (get_key_color($EVENT_NOTE) = $KEY_COLOR_RED)
      message("Bravo!")

      set_key_color($random_key,$KEY_COLOR_INACTIVE)
      $random_key := random(60,71)
      set_key_color($random_key,$KEY_COLOR_RED)
   else
      message("Try again!")
   end if
end on

on release
   message("")
end on
```

catch me if you can

See Also

`set_key_color()`

get_key_name()

```
get_key_name(<note-nr>)
```

returns the name of the specified key

Examples

```
on init

    declare $count
    while ($count < 128)
        set_key_name($count,"")
        inc($count)
    end while

   set_key_name(60,"Middle C")

end on

on note
   message(get_key_name($EVENT_NOTE))
end on
```

See Also

set_key_name()

get_key_triggerstate()

```
get_key_triggerstate(<note-nr>)
```

returns the pressed state of the specified note number (i.e. key) on the KONTAKT keyboard, can be either 1 (key pressed) or 0 (key released)

Remarks

`get_key_triggerstate()` works only with `set_key_pressed_support()` set to 1.

Examples

```
on init
    set_key_pressed_support(1)
end on
on note
    set_key_pressed($EVENT_NOTE,1)
    message(get_key_triggerstate($EVENT_NOTE))
end on
on release
    set_key_pressed($EVENT_NOTE,0)
    message(get_key_triggerstate($EVENT_NOTE))
end on
```

See Also

`set_key_pressed()`
`set_key_pressed_support()`

get_key_type()

```
get_key_type(<note-nr>)
```

returns the key type constant of the specified key.

See Also

`set_key_type()`

get_keyrange_min_note()

```
get_keyrange_min_note(<note-nr>)
```

returns the lowest note of the specified key range

Remarks

Since a key range cannot have overlapping notes, it is sufficient with all `get_keyrange_xxx()` commands to specify the key range with one note number only.

Examples

```
on init

    declare $count
    while ($count < 128)

        remove_keyrange($count)
        inc($count)
    end while

    set_keyrange(36,72,"Middle Range")

end on

on note
   message(get_keyrange_min_note($EVENT_NOTE))
end on
```

See Also

`set_keyrange()`

get_keyrange_max_note()

```
get_keyrange_max_note(<note-nr>)
```

returns the highest note of the specified key range

Remarks

Since a key range cannot have overlapping notes, it is sufficient with all `get_keyrange_xxx()` commands to specify the key range with one note number only.

Examples

```
on init

    declare $count
    while ($count < 128)

        remove_keyrange($count)
        inc($count)
    end while

    set_keyrange(36,72,"Middle Range")

end on

on note
   message(get_keyrange_min_note($EVENT_NOTE))
end on
```

See Also

`set_keyrange()`

get_keyrange_name()

```
get_keyrange_name(<note-nr>)
```

returns the name of the specified key range

Remarks

Since a key range cannot have overlapping notes, it is sufficient with all `get_keyrange_xxx()` commands to specify the key range with one note number only.

Examples

```
on init

    declare $count
    while ($count < 128)

        remove_keyrange($count)
        inc($count)
    end while

    set_keyrange(36,72,"Middle Range")

end on

on note
   message(get_keyrange_name($EVENT_NOTE))
end on
```

See Also

`set_keyrange()`

set_key_color()

```
set_key_color(<note-nr>,<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_RED
$KEY_COLOR_ORANGE
$KEY_COLOR_LIGHT_ORANGE
$KEY_COLOR_WARM_YELLOW
$KEY_COLOR_YELLOW
$KEY_COLOR_LIME
$KEY_COLOR_GREEN
$KEY_COLOR_MINT
$KEY_COLOR_CYAN
$KEY_COLOR_TURQUOISE
$KEY_COLOR_BLUE
$KEY_COLOR_PLUM
$KEY_COLOR_VIOLET
$KEY_COLOR_PURPLE
$KEY_COLOR_MAGENTA
$KEY_COLOR_FUCHSIA

$KEY_COLOR_DEFAULT (sets the key to KONTAKT's standard color for mapped notes)
$KEY_COLOR_INACTIVE (resets the key to standard black and white)
$KEY_COLOR_NONE (resets the key to its normal KONTAKT color, e.g. red for internal keyswitches)

Remarks

The keyboard colors reside outside of KSP, i.e. changing the color of a key is similar to changing a KONTAKT knob with `set_engine_par()`. It is therefore a good practice to set all keys to either $KEY_COLOR_INACTIVE or $KEY_COLOR_NONE in the init callback or whenever changed later.

Example

(see next page)

```
on init
    message("")
    declare ui_button $Color

    declare $count
    declare $note_count
    declare $color_count
    declare %white_keys[7] := (0,2,4,5,7,9,11)
    declare %colors[16] := (...
    $KEY_COLOR_RED,$KEY_COLOR_ORANGE,$KEY_COLOR_LIGHT_ORANGE,...
    $KEY_COLOR_WARM_YELLOW,$KEY_COLOR_YELLOW,$KEY_COLOR_LIME,...
    $KEY_COLOR_GREEN,$KEY_COLOR_MINT,$KEY_COLOR_CYAN,...
    $KEY_COLOR_TURQUOISE,$KEY_COLOR_BLUE,$KEY_COLOR_PLUM,...
    $KEY_COLOR_VIOLET,$KEY_COLOR_PURPLE,$KEY_COLOR_MAGENTA,$KEY_COLOR_FUCHSIA)

    $count := 0
    while ($count < 128)
        set_key_color($count,$KEY_COLOR_NONE)
        inc($count)
    end while
end on

on ui_control ($Color)
    if ($Color = 1)

        $count := 0
        while ($count < 128)
            set_key_color($count,$KEY_COLOR_INACTIVE)
            inc($count)
        end while

        $note_count := 0
        $color_count := 0
        while ($color_count < 16)

            if (search(%white_keys,(60 + $note_count) mod 12) # -1)
                set_key_color(60 + $note_count,%colors[$color_count])
                inc ($color_count)
            end if

            inc($note_count)

        end while

    else

        $count := 0
        while ($count < 128)
            set_key_color($count,$KEY_COLOR_NONE)
            inc($count)
        end while

    end if

end on
```

KONTAKT rainbow

See Also

```
set_control_help()
get_key_color()
set_key_name()
set_keyrange()
```

set_key_name()

```
set_key_name(<note-nr>,<name>)
```

assigns a text string to the specified key

Remarks

Key names are instrument parameters and reside outside KSP, i.e. changing the key name is similar to changing a KONTAKT knob with `set_engine_par()`. Make sure to always reset all key names in the init callback or whenever changed later.

Key names and ranges are displayed in KONTAKT's info pane when hovering the mouse over the key on the KONTAKT keyboard.

Examples

```
on init

    declare $count
    while ($count < 128)
        set_key_name($count,"")
        inc($count)
    end while

   set_key_name(60,"Middle C")

end on
```

See Also

`set_keyrange()`
`get_key_name()`

set_key_pressed()

```
set_key_pressed(<note-nr>,<value>)
```

sets the trigger state of the specified key on KONTAKT's keyboard either to pressed/on (1) or released/off (0)

Remarks

By using `set_key_pressed()` in combination with `set_key_pressed_support()` it is possible to show script generated notes on KONTAKT's keyboard. The typical use case would be if an instrument features an in-built sequencer/harmonizer and the triggered notes should be shown on the keyboard.

Examples

```
on init
    set_key_pressed_support(1)
end on
on note
    set_key_pressed($EVENT_NOTE,1)
end on
on release
    set_key_pressed($EVENT_NOTE,0)
end on
```

insert this after an arpeggiator or harmonizer script

See Also

`set_key_pressed_support()`
`get_key_triggerstate()`

set_key_pressed_support()

`set_key_pressed_support(<mode>)`

sets the pressed state support mode for KONTAKT"S keyboard. The available modes are:

0: KONTAKT handles all pressed states, `set_key_pressed()` commands are ignored (default mode)
1: KONTAKT's keyboard is only affected by `set_key_pressed()` commands

Remarks

The pressed state mode resides outside KSP, i.e. changing the mode is similar to changing a KONTAKT knob with `set_engine_par()`. Make sure to always set the desired mode in the init callback.

Examples

```
on init
    declare ui_button $Enable
    set_key_pressed_support(0)
end on

on ui_control ($Enable)
   set_key_pressed_support($Enable)
end on

on note
   play_note($EVENT_NOTE+4,$EVENT_VELOCITY,0,-1)
   play_note($EVENT_NOTE+7,$EVENT_VELOCITY,0,-1)
   set_key_pressed($EVENT_NOTE,1)
   set_key_pressed($EVENT_NOTE+4,1)
   set_key_pressed($EVENT_NOTE+7,1)
end on

on release
   set_key_pressed($EVENT_NOTE,0)
   set_key_pressed($EVENT_NOTE+4,0)
   set_key_pressed($EVENT_NOTE+7,0)
end on
```

press the button and you'll see what you hear

See Also

`set_key_pressed()`
`get_key_triggerstate()`

set_key_type()

```
set_key_type(<note-nr>,<key-type-constant>)
```

assigns a key type to the specified key.

The following key types are available:
`$NI_KEY_TYPE_DEFAULT` (i.e. normal mapped notes that produce sound)
`$NI_KEY_TYPE_CONTROL` (i.e. key switches or other notes that do not produce sound)
`$NI_KEY_TYPE_NONE` (resets the key to its normal KONTAKT behaviour)

Remarks

Setting the key type is useful for supported hosts like KOMPLETE KONTROL, where keys with control functionality (e.g. key switches) should not be affected by any note processing.

Examples

```
on init

    declare $count

    $count := 0
    while ($count < 128)
        set_key_type($count,$NI_KEY_TYPE_NONE)
        inc($count)
    end while

    $count := 36
    while ($count <= 96)

        select ($count)

            case 36 to 47 {e.g. key switch}
                set_key_type($count,$NI_KEY_TYPE_CONTROL)

            case 48 to 96 {e.g. main notes}
                set_key_type($count,$NI_KEY_TYPE_DEFAULT)

        end select

        inc($count)
    end while
end on
```

See Also

`get_key_type()`

set_keyrange()

```
set_keyrange(<min-note>,<max-note>,<name>)
```

assigns a text string to the specified range of keys.

Remarks

Key ranges are instrument parameters and reside outside KSP, i.e. changing the key range is similar to changing a KONTAKT knob with `set_engine_par()`. Make sure to always remove all key ranges in the init callback or whenever changed later.

There can be up to 16 key ranges per instrument.

Key names and ranges are displayed in KONTAKT's info pane when hovering the mouse over the key on the KONTAKT keyboard. The range name is followed by the key name (separated by a dash).

Examples

```
on init

    declare $count
    while ($count < 128)

        remove_keyrange($count)
        inc($count)
    end while

    set_keyrange(36,72,"Middle Range")

end on
```

See Also

`remove_keyrange()`
`set_key_name()`

remove_keyrange()

```
remove_keyrange(<note-nr>)
```

assigns a text string to the specified range of keys

Remarks

Key ranges are instrument parameters and reside outside KSP, i.e. changing the key range is similar to changing a KONTAKT knob with `set_engine_par()`. Make sure to always remove all key ranges in the init callback or whenever changed later.

Examples

```
on init

    declare $count
    while ($count < 128)

        remove_keyrange($count)
        inc($count)
    end while

    set_keyrange(36,72,"Middle Range")

end on
```

See Also

`set_keyrange()`

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 $grp_idx
   $grp_idx := 0

   declare $env_idx
   $env_idx := find_mod(0,"VOL_ENV")

   declare ui_knob $Attack (0,1000000,1)
   set_knob_unit($Attack,$KNOB_UNIT_MS)

   $Attack := get_engine_par($ENGINE_PAR_ATTACK,$grp_idx,$env_idx,-1)

   set_knob_label($Attack,get_engine_par_disp...
   ($ENGINE_PAR_ATTACK,$grp_idx,$env_idx,-1))

end on
on ui_control ($Attack)

   set_engine_par($ENGINE_PAR_ATTACK,$Attack,$grp_idx,$env_idx,-1)

   set_knob_label($Attack,get_engine_par_disp...
   ($ENGINE_PAR_ATTACK,$grp_idx,$env_idx,-1))

end on
```

controlling the attack time of the volume envelope of the first group. Note: the envelope has been manually renamed to "VOL_ENV"

```
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 controls the velocity to volume modulation intensity of all groups

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 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 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>)	
returns the value of a specific engine parameter	
<parameter>	specifies the engine parameter by using one of the built in engine parameter variables
<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 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 $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>)	
returns the displayed string 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 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 $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

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)

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

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, for example $ENGINE_PAR_CUTOFF
<value>	The value to which the specified parameter is set. The range of values is always 0 to 1000000, except for switches in which case it is 0 or 1.
<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

Load/Save Commands

General Information

File Formats

It is possible to load and save the following file formats:

- KONTAKT arrays (.nka files)
- MIDI files (.mid) to be used with the MIDI file commands in KSP
- IR samples (.wav) to be used with KONTAKT's convolution effect (loading only)

Async Handling

Loading and saving files cannot be executed in real time. This is why all load/save commands return a unique value upon completion of their action. You can use this value in combination with `$NI_ASYNC_ID` and `$NI_ASYNC_EXIT_STATUS` within the `on_async_complete` callback to check wether the the command has completed ist action, and wether or not the loading or saving was succesful.

Path Handling

All file paths in KSP use a slash character (/) as a folder separator (backslash characters are not supported). The full path has to start with a slash character"/"

Examples:

factory folder on OS X:
```
/Library/Application Support/Native Instruments/Kontakt 5/
```

factory folder on Windows:
```
/C:/Program Files/Common Files/Native Instruments/Kontakt 5/
```

When loading or saving files with an absolute path as opposed to loading from the Resource Container, always use path variables in combination with `get_folder()`.

See Also

```
$NI_ASYNC_ID
$NI_ASYNC_EXIT_STATUS
on async_complete
```

get_folder()

get_folder(<path-variable>)	
returns the path specified with the built-in path variable	
<path-variable>	the following path variables are available: $GET_FOLDER_LIBRARY_DIR if used with an nki belonging to an encoded library: library folder if used with an unencoded nki: the user content directory $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

- The behaviour $GET_FOLDER_LIBRARY_DIR changed from KONTAKT 5 on. If the nki belongs to an encoded library, it will point to its library folder. Otherwise, the user content directory is returned.

Examples

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

displaying the path of the factory folder of KONTAKT

See Also

load_ir_sample()
$GET_FOLDER_LIBRARY_DIR
$GET_FOLDER_FACTORY_DIR
$GET_FOLDER_PATCH_DIR

load_array()

load_array(<array-variable>,<mode>)	
loads an array from an external file (.nka file)	
<array-variable> <mode>	the array variable, this name must be present in the .nka file 0: A dialog window pops up, allowing you to select an .nka file. Can only be used in ui, pgs and persistence_changed callbacks. 1: The array is directly loaded from the "Data" folder. For user instruments, the "Data" folder is located beside the resource container. For library instruments, the "Data" folder is located here: OS X: <UserName>/Library/Application Support/<Library Name>/ Win: C:\User\<UserName>\AppData\Local\<Library Name>\ Can be used in ui, pgs, init (synchronous) and persistence_changed callbacks. 2: The array is directly loaded from the "data" folder **inside** the resource container. Can be used in ui, pgs, init (synchronous) and persistence_changed callbacks.

Remarks

- It is also possible to load string arrays from .nka files.
- It is not possible to load an array with %xyz in its .nka file into array %abc.
- The array data is not directly available after the load_array() command has been executed since the command works asynchronous. The only situation in which the values are instantly available is when using mode 1 or mode 2 within an init callback.
- When using mode 0 the callback continues even if the loading dialog is still open.
- Mode 2 is only available for loading arrays, i.e. save_array() does not have this option.
- 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.

Examples

(see next page)

```
on init
   declare $count
   declare ui_button $Load
   declare ui_button $Save
   declare ui_table %table[8] (2,2,100)
   make_persistent(%table)
   declare %preset[8]
   declare $load_arr_id
   $load_arr_id := -1
   declare $save_arr_id
   $save_arr_id := -1
end on

on ui_control (%table)
   $count := 0
   while($count < 8)
      %preset[$count] := %table[$count]
      inc($count)
   end while
end on

on ui_control ($Load)
   $load_arr_id := load_array(%preset,0)
end on

on ui_control ($Save)
   $save_arr_id := save_array(%preset,0)
end on

on async_complete
   if ($NI_ASYNC_ID = $load_arr_id)
      $load_arr_id := -1
      $Load := 0
      if ($NI_ASYNC_EXIT_STATUS = 1)
         $count := 0
         while($count < 8)
            %table[$count] := %preset[$count]
            inc($count)
         end while
      end if
   end if
   if ($NI_ASYNC_ID = $save_arr_id)
      $save_arr_id := -1
      $Save := 0
   end if
end on
```

Exporting and loading the contents of a ui table

See Also

$NI_ASYNC_ID
$NI_ASYNC_EXIT_STATUS
on async_complete
save_array()

load_array_str()

load_array_str(<array-variable>,<path>)	
loads an array from an external file (.nka file) using the file's absolute path	
<array-variable>	the array variable, this name must be present in the .nka file
<path>	the absolute path of the .nka file

Remarks

- The behaviour is similar to load_array()with mode set to 0, but instead of manually choosing an .nka file you can specify it with an absolute path.
- Can be used in init (synchronous), persistence_changed, ui and pgs callbacks.

Examples

(see next page)

```
on init
   set_ui_height(2)

   declare @basepath_browser
   {set browser path here, for example
   @basepath_browser := "/Users/<username>/Desktop/Arrays"}

   declare @file_path
   make_persistent(@file_path)

   declare @file_name
   make_persistent(@file_name)

   declare ui_file_selector $file_browser
   declare $browser_id
   $browser_id := get_ui_id($file_browser)
   set_control_par_str($browser_id,$CONTROL_PAR_BASEPATH,@basepath_browser)
   set_control_par($browser_id,$CONTROL_PAR_WIDTH,112)
   set_control_par($browser_id,$CONTROL_PAR_HEIGHT,68)
   set_control_par($browser_id,$CONTROL_PAR_COLUMN_WIDTH,110)
   set_control_par($browser_id,$CONTROL_PAR_FILE_TYPE,$NI_FILE_TYPE_ARRAY)
   move_control_px($file_browser,66,2)

   declare ui_table %table[8] (2,2,100)
   make_persistent(%table)
   move_control(%table,3,1)

   declare %preset[8]

   declare $load_arr_id
   $load_arr_id := -1
   declare $count
end on

on async_complete

   if ($NI_ASYNC_ID = $load_arr_id)

      $load_arr_id := -1

      if ($NI_ASYNC_EXIT_STATUS = 0)
         message("Array not found!")
      else
         message("")
         $count := 0
         while($count < 8)
            %table[$count] := %preset[$count]
            inc($count)
         end while
      end if
   end if
end on

on ui_control ($file_browser)
   @file_name := fs_get_filename($browser_id,0)
   @file_path := fs_get_filename($browser_id,2)
   $load_arr_id := load_array_str(%preset,@file_path)
end on
```

Loading different table presets with a browser – make sure to first set the browser path of the file selector to point to a folder with compatible .nka files

load_ir_sample()

load_ir_sample(<file-path>,<slot>,<generic>)	
loads an impulse response sample into KONTAKT's convolution effect	
<file-path>	the absolute file path of the IR sample If no path is specified, the command will look for the specified sample within the “ir_samples” folder of the Resource Container. If no Resource Container is available, the folder "ir_samples" within the KONTAKT user folder will be checked. The KONTAKT user folder is loacated here: OS X /Users/<username>/Documents/Native Instruments/Kontakt 5/ Windows C:/Users/<username>/Documents/Native Instruments/Kontakt 5/
<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 For busses, this parameter specifies the actual bus: $NI_BUS_OFFSET + [0-15] one of the 16 busses

Remarks

- 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.

Examples

(see next page)

```
on init
   declare ui_button $Load
   declare $load_ir_id
   $load_ir_id := -1
end on

on ui_control ($Load)
   $load_ir_id := load_ir_sample("Small Ambience.wav",0,0)
   $Load := 0
end on

on async_complete

   if ($NI_ASYNC_ID = $load_ir_id)

      $load_ir_id := -1

      if ($NI_ASYNC_EXIT_STATUS = 0)
         message("IR sample not found!")
      else
         message("IR sample loaded!")
      end if

   end if

end on
```

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

See Also

$NI_ASYNC_ID
get_folder()
on async_complete

save_array()

save_array(<array-variable>,<mode>)	
saves an array to an external file (i.e. an .nka file)	
<array-variable> <mode>	the array to be saved 0: A dialog window pops up, allowing you to save the .nka file. Can only be used in ui, pgs and persistence_changed callbacks. 1: The array is directly loaded from the "Data" folder. For user instruments, the "Data" folder is located beside the resource container. For library instruments, the "Data" folder is located here: OS X: <UserName>/Library/Application Support/<Library Name>/ Win: C:\User\<UserName>\AppData\Local\<Library Name>\ Can be used in ui, pgs, and persistence_changed callbacks.

Remarks

- It is also possible to save string arrays into .nka files.
- The exported .nka file consists of the name of the array followed its values.
- When using mode 0 the callback continues even if the loading dialog is still open.

See Also

```
$NI_ASYNC_ID
$NI_ASYNC_EXIT_STATUS
on async_complete
load_array()
```

save_array_str()

```
save_array_str(<array-variable>,<path>)
```

saves an array to an external file (i.e. an .nka file), using the specified absolute path

`<array-variable>`	the array to be saved
`<path>`	the absolute path of the .nka file to be saved

Remarks

- The behaviour is similar to `save_array()`, but instead of manually choosing a save location, you can directly save the file to the specified location.
- If the file does not exist (but the folder does), a new .nka file will be created
- Can be used in persistence_changed, ui and pgs callbacks.

Examples

{see next page}

```
on init
   declare $count

   declare @path
   {set save path here, for example
   @path := "/Users/<username>/Desktop/Arrays/"}

   declare ui_button $Save

   declare ui_table %table[8] (2,2,100)
   make_persistent(%table)

   declare %preset[8]

   declare $save_arr_id
   $save_arr_id := -1

   declare ui_text_edit @preset_name
   make_persistent(@preset_name)

   set_control_par_str(get_ui_id(@preset_name),$CONTROL_PAR_TEXT,"empty")
   set_control_par(get_ui_id(@preset_name),$CONTROL_PAR_FONT_TYPE,25)
   set_control_par(get_ui_id(@preset_name),$CONTROL_PAR_POS_X,73 + 3*92)
   set_control_par(get_ui_id(@preset_name),$CONTROL_PAR_POS_Y,2)

   declare ui_label $pattern_lbl(1,1)
   set_text($pattern_lbl,"")
   move_control_px($pattern_lbl,66 + 3*92,2)

end on

on ui_control (%table)
   $count := 0
   while($count < 8)
      %preset[$count] := %table[$count]
      inc($count)
   end while
end on

on ui_control ($Save)
   $save_arr_id := save_array_str(%preset,@path & @preset_name & ".nka")
end on

on async_complete
   if ($NI_ASYNC_ID = $save_arr_id)
      $save_arr_id := -1
      $Save := 0
   end if
end on
```

Save table presets with custom names – make sure to set the path where the .nka files will be saved

See Also

```
save_array()
load_array_str()
```

save_midi_file()

save_midi_file(<path>)	
saves a MIDI file with a range specified by the `mf_set_export_area()` command.	
`<path>`	the absolute path of the MIDI file

Example

```
on init
   declare @path
   {set save path here, for example
   @path := "/Users/<username>/Desktop/MIDI Files/"}

   declare ui_text_edit @file_name
   set_control_par_str(get_ui_id(@file_name),$CONTROL_PAR_TEXT,"<empty>")
   set_control_par(get_ui_id(@file_name),$CONTROL_PAR_FONT_TYPE,25)
   make_persistent(@file_name)
   move_control_px(@file_name,73,2)

   declare ui_label $file_name_lbl(1,1)
   set_text($file_name_lbl,"")
   move_control_px($file_name_lbl,66,2)

   declare ui_button $Save
   move_control($Save,2,1)

   declare $save_mf_id
   $save_mf_id := -1

end on

on ui_control ($Save)
   $save_mf_id := save_midi_file(@path & @file_name & ".mid")
end on

on async_complete
   if ($NI_ASYNC_ID = $save_mf_id)
      $save_mf_id := -1
      $Save := 0
   end if
end on
```

Saving a MIDI file

See Also

`mf_insert_file()`
`mf_set_export_area()`

MIDI Object Commands

General Information

Please note that in KONTAKT version 5.2, the MIDI file handling has been significantly updated. Commands and working methods from before the 5.2 release will remain in order to keep backwards compatibility; however this reference will document the post 5.2 working method.

You can only use one MIDI object at a time within an NKI. The MIDI object is held in memory and can be accessed by any of the script slots. It is possible to add, remove and edit MIDI events within the object, as well as import and export MIDI files.

The Multi Script can also hold one MIDI object, and handles it in the same way as an NKI.

Creating, Importing and Exporting MIDI files

When you initialize an instrument, an empty MIDI object is initialized with it. You can either start editing the object by defining a buffer size and inserting events, or by inserting a whole MIDI file.

If you want to create a MIDI sequence from scratch, you first need to assign a buffer size, which effectively creates a number of inactive MIDI events. From this point you can activate (ie. insert) and edit MIDI events using the MIDI event commands.

You can also load a MIDI file to use or edit the data in a script. Depending on the command and variables you use, this will either be combined with any existing MIDI data, or will replace the existing data. It should be noted that loading a MIDI file is an asynchronus command, and thus the common asynchronus loading commands and working methods apply.

MIDI objects can be exported from KONTAKT either by using the `save_midi_file()` command, or via a drag and drop enabled label element. In either case, it is possible to define the export area, both in terms of start and end times, as well as the start and end tracks, by using the `mf_set_export_area()` command.

Navigation and Editing

MIDI events in KONTAKT's MIDI object are given event parameters, which are accessed using either the `mf_get_event_par()` or `mf_set_event_par()` commands. A unique event ID can be used to access a specific event, or you can navigate through events by position. The event ID is assigned whenever a MIDI event is created or loaded.

In order to access the event data of a loaded MIDI file, you can navigate around the MIDI events with a position marker, something analogous to a play-head. The position marker will focus on one single event at a time, allowing you to use a variety of commands to access or edit the event's parameters. You have the option to either navigate from one event to the next, or to specify exact positions in MIDI ticks.

It should be noted that MIDI note off messages are not used. When you load a MIDI file using the `mf_insert_file()` command, the note off events are used to give a length parameter to the respective note on event, and are then discarded.

mf_insert_file()

mf_insert_file(<path>,<track-offset>,<position-offset>,<mode>)	
inserts a MIDI file into the MIDI object.	
<path>	the absolute path of the MIDI file, including the file name
<track-offset>	applies a track offset to the MIDI data
<position-offset>	applies a position offset, in ticks, to the MIDI data
<mode>	defines the mode of insertion: 0: replace all existing events 1: replace only overlapping events 2: merge all events

Remarks

- The loading of MIDI files with this command is asynchronous, so it is advised to use the `async_complete` callback to check the status of the load. However, the `async_complete` callback will not be called if this command is used in the init callback.
- This command will pair Note On and Note Off events to a single Note On with a Note Length parameter. The Note Off events will be discarded.

Example

(see next page)

```
on init
   declare @file_name
   declare @filepath

   @file_name := "test.mid"
   @filepath := get_folder($GET_FOLDER_FACTORY_DIR) & @file_name

   declare $load_mf_id
   declare ui_button $load_file
end on

on ui_control($load_file)
   $load_mf_id := mf_insert_file(@filepath,0,0,0)
end on

on async_complete
   if ($NI_ASYNC_ID = $load_mf_id)

      $load_mf_id := -1

      if ($NI_ASYNC_EXIT_STATUS = 0)
         message("FATAL ERROR: MIDI file not found!")
      else
         message("Loaded MIDI File: " & @file_name)
      end if
   end if
end on
```

Loading a MIDI file with a button – in order for this to work you'll have to put a MIDI file called "test.mid" into your KONTAKT Factory folder. Otherwise the defined error message will be displayed.

See Also

$NI_ASYNC_ID
$NI_ASYNC_EXIT_STATUS
on async_complete
save_midi_file()
mf_set_event_par()
mf_get_event_par()

mf_set_export_area()

mf_set_export_area(<name>,<start-pos>,<end-pos>,<start-track>,<end-track>)	
defines the part of the MIDI object that will be exported when using a drag and drop area, or the save_midi_file() command.	
<name>	sets the name of the exported file
<start-pos>	defines the start position (in ticks) of the export area. Use -1 to set this to the start of the MIDI object.
<end-pos>	defines the end position (in ticks) of the export area. Use -1 to set this to the end of the MIDI object.
<start-track>	defines the first track to be included in the export area. Use -1 to set this to the first track of the MIDI object.
<end-track>	defines the last track to be included in the export area. Use -1 to set this to the last track of the MIDI object.

Remarks

- If a start point is given a value greater than the end point, the values will be swapped.
- When this command is executed, the events in the range are checked if they are valid MIDI commands. The command will return a value of 0 if all events are valid, otherwise it will return the event ID of the first invalid event.

Example

```
on init
   @filepath := get_folder($GET_FOLDER_FACTORY_DIR) & "test.mid"
   mf_insert_file(@filepath,0,0,0)

   declare ui_button $check_area
   declare $area_status
end on

on ui_control($check_area)
   $area_status := mf_set_export_area(“name”,-1,-1,-1,-1)
   if($area_status = 0)
   message(“All Good”)
   else
   message(“Error: check event with ID ” & $area_status)
   end if
end on
```

A simple script, using this command to check if all events in a MIDI file are valid. If there is an error it will display the event ID of the first invalid event – in order for this to work you'll have to put a MIDI file called "test.mid" into your KONTAKT Factory folder.

See Also

```
mf_insert_file()
$CONTROL_PAR_DND_BEHAVIOUR
save_midi_file()
```

mf_set_buffer_size()

```
mf_set_buffer_size(<size>)
```

defines a number of inactive MIDI events, that can be activated and edited

`<size>`	the size of the MIDI object edit buffer

Remarks

- Using the `mf_insert_event()` and `mf_remove_event()` technically activate or deactivate events in the buffer.
- It is not possible to insert MIDI events without first setting a buffer size
- The maximum buffer size is 1,000,000 events (including both active and inactive events)
- If this command is called outside of the init callback, it is asynchronous, and thus calls the `async_complete` callback.
- Inserting a MIDI event will decrease the buffer size by one. Removing an event will increase it by one.
- Inserting a MIDI file will not affect the buffer.

See Also

```
mf_insert_file()
mf_get_buffer_size()
mf_reset()
mf_insert_event()
mf_remove_event()
save_midi_file()
```

mf_get_buffer_size()

```
mf_get_buffer_size()
```

returns the size of the MIDI event buffer

Remarks

- The maximum buffer size is 1,000,000 events (including both active and inactive events)
- Inserting a MIDI event will decrease the buffer size by one. Removing an event will increase it by one.

See Also

```
mf_insert_file()
mf_set_buffer_size()
mf_reset()
mf_insert_event()
mf_remove_event()
save_midi_file()
```

mf_reset()

```
mf_reset()
```

resets the MIDI object, sets the event buffer to zero, and removes all events

Remarks

- This command purges all MIDI data, use with caution
- This command is also asynchronous, and thus calls the `async_complete` callback

See Also

```
mf_insert_file()
mf_set_buffer_size()
mf_reset()
mf_insert_event()
mf_remove_event()
save_midi_file()
```

mf_insert_event()

mf_insert_event(<track>,<pos>,<command>,<byte1>,<byte2>)	
activates an inactive MIDI event in the MIDI object. However, because the command and position are defined in this command, it can be considered as an insertion.	
<track>	the track into which the MIDI event will be inserted
<pos>	the position at which the event will be inserted, in MIDI ticks
<command>	defines the command type of the MIDI event, can be one of the following: $MIDI_COMMAND_NOTE_ON $MIDI_COMMAND_POLY_AT $MIDI_COMMAND_CC $MIDI_COMMAND_PROGRAM_CHANGE $MIDI_COMMAND_MONO_AT $MIDI_COMMAND_PITCH_BEND
<byte1>	the first byte of the MIDI command
<byte2>	the second byte of the MIDI command

Remarks

- It is not possible to insert MIDI events without first setting an event buffer size with the `mf_set_buffer_size()` command
- Using this command when the buffer is full (i.e. has a size of zero) will do nothing
- You can retrieve the event ID of the inserted event in a variable by writing:
  `<variable> := mf_insert_event(<track>,<pos>,<command>,<byte1>,<byte2>)`

See Also

```
mf_insert_file()
mf_set_buffer_size()
mf_get_buffer_size()
mf_reset()
mf_remove_event()
save_midi_file()
```

mf_remove_event()

mf_remove_event(<event-id>)	
deactivates an event in the MIDI object, effectively removing it	
<event-id>	the ID of the event to be deactivated

Remarks

- Using this command will increase the MIDI event buffer size by one

See Also

mf_insert_file()
mf_set_buffer_size()
mf_get_buffer_size()
mf_reset()
mf_insert_event()
save_midi_file()

mf_set_event_par()

mf_set_event_par(<event-id>,<parameter>,<value>)	
sets an event parameter	
<event-id>	the ID of the event to be edited
<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 MIDI event: $EVENT_PAR_MIDI_CHANNEL $EVENT_PAR_MIDI_COMMAND $EVENT_PAR_MIDI_BYTE_1 $EVENT_PAR_MIDI_BYTE_2 $EVENT_PAR_POS $EVENT_PAR_NOTE_LENGTH $EVENT_PAR_TRACK_NR
<value>	the value of the event parameter

Remarks

- You can control all events in the MIDI object by using the $ALL_EVENTS constant as the event ID.
- You can access the currently selected event by using the $CURRENT_EVENT constant.
- You can also control events by track, or group them with markers by using the by_track() and by_mark() commands.

See Also

mf_insert_file()
mf_insert_event()
mf_remove_event()
$ALL_EVENTS
$CURRENT_EVENT
by_marks()
by_track()
mf_set_mark()
mf_get_id()
save_midi_file()

mf_get_event_par()

mf_get_event_par(<event-id>,<parameter>)	
returns the value of an event parameter	
<event-id>	the ID of the event to be edited
<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 MIDI event: $EVENT_PAR_MIDI_CHANNEL $EVENT_PAR_MIDI_COMMAND $EVENT_PAR_MIDI_BYTE_1 $EVENT_PAR_MIDI_BYTE_2 $EVENT_PAR_POS $EVENT_PAR_NOTE_LENGTH $EVENT_PAR_ID $EVENT_PAR_TRACK_NR

Remarks

- You can access all events in the MIDI object by using the $ALL_EVENTS constant as the event ID.
- You can access the currently selected event by using the $CURRENT_EVENT constant.
- You can also access events by track, or group them with markers by using the by_track() and by_mark() commands.

See Also

mf_insert_file()
mf_insert_event()
mf_remove_event()
$CURRENT_EVENT
mf_get_id()
save_midi_file()

mf_get_id()

```
mf_get_id()
```

returns the ID of the currently selected event (when using the navigation commands like `mf_get_first()`, and `mf_get_next()`, etc)

See Also

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

mf_set_mark()

mf_set_mark(<event-id>,<mark>,<status>)	
marks an event, so that you may groups events together and process that group quickly	
<event-id>	the ID of the event to be marked
<mark>	the mark number. Use the constants $MARK_1 to $MARK_10
<status>	set this to 1 to mark an event or to 0 to unmark an event

See Also

mf_insert_file()
mf_insert_event()
mf_remove_event()
$ALL_EVENTS
$CURRENT_EVENT
mf_get_mark()
by_marks()
by_track()
mf_get_mark()
mf_get_id()
save_midi_file()

mf_get_mark()

mf_get_mark(<event-id>,<mark>)	
checks if an event is marked or not. Returns 1 if it is marked, or 0 if it is not.	
<event-id>	the ID of the event to be edited
<mark>	the mark number. Use the constants $MARK_1 to $MARK_10

See Also

mf_insert_file()
mf_insert_event()
mf_remove_event()
$ALL_EVENTS
$CURRENT_EVENT
mf_set_mark()
by_marks()
by_track()
mf_get_mark()
mf_get_id()
save_midi_file()

by_marks()

by_marks(<mark>)	
can be used to access a user defined group of events	
<mark>	the mark number. Use the constants $MARK_1 to $MARK_10

See Also

mf_insert_file()
mf_insert_event()
mf_remove_event()
$ALL_EVENTS
$CURRENT_EVENT
mf_set_mark()
mf_get_mark()
by_marks()
by_track()
mf_get_mark()
mf_get_id()
save_midi_file()

by_track()

by_track(<track>)	
can be used to group events by their track number	
<track>	the track number of the events you wish to access

Remarks

- Similar in functionality as the by_marks() command

See Also

mf_insert_file()
mf_insert_event()
mf_remove_event()
$ALL_EVENTS
$CURRENT_EVENT
mf_set_mark()
mf_get_mark()
by_marks()
mf_get_mark()
mf_get_id()
save_midi_file()

mf_get_first()

mf_get_first(<track-index>)	
moves the position marker 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

- Using this command will also select the event at the position marker for editing.

See Also

```
mf_insert_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_last()

mf_get_last(<track-index>)	
moves the position marker 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

- Using this command will also select the event at the position marker for editing.

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_next()

mf_get_next(<track-index>)	
moves the position marker 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

- Using this command will also select the event at the position marker for editing.

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>)	
moves the position marker to the next event in the MIDI track right after the defined position.	
<track-index> <pos>	the number of the track you want to edit. **-1** refers to the whole file. position in ticks

Remarks

- Using this command will also select the event at the position marker for editing.

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_prev()

mf_get_prev(<track-index>)	
moves the position marker 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

- Using this command will also select the event at the position marker for editing.

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>)	
moves the position marker to the first event before the defined position	
<track-index>	the number of the track you want to edit. **-1** refers to the whole file.
<pos>	position in ticks

Remarks

- Using this command will also select the event at the position marker for editing.

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_get_num_tracks()

```
mf_get_num_tracks()
```

returns the number of tracks in a MIDI object.

See Also

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

Built-in Variables

General

$ALL_GROUPS

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

$ALL_EVENTS

addresses all events in functions which deal with an event ID number
this constant also works with MIDI event commands that require a MIDI event ID

Bit Mark Constants

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

$MARK_1
$MARK_2
…
$MARK_28

%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 by 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. The engine uptime is calculated from the sample rate and can thus be used in 'musical' contexts (eg. building arpeggiators or sequencers) as it remains in sync, even in an offline bounce.

$EVENT_ID

unique ID number of the event which triggered the callback

$CURRENT_EVENT

the currently selected MIDI event (i.e. the MIDI event at the position marker)

$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()` and `get_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_PAR_ZONE_ID (returns the zone id of the event – can only be used with active events; returns -1 if no zone is triggered; returns the highest zone id if more than one zone is triggered by the event, make sure the voice is running by writing e.g. `wait(1)` before retrieving the zone ID)

MIDI Event Parameter Constants

event parameters to be used with mf_set_event_par() and mf_get_event_par()

```
$EVENT_PAR_0
$EVENT_PAR_1
$EVENT_PAR_2
$EVENT_PAR_3
$EVENT_PAR_MIDI_CHANNEL
$EVENT_PAR_MIDI_COMMAND
$EVENT_PAR_MIDI_BYTE_1
$EVENT_PAR_MIDI_BYTE_2
$EVENT_PAR_POS
$EVENT_PAR_NOTE_LENGTH
$EVENT_PAR_ID
$EVENT_PAR_TRACK_NR
```

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.
the size of the array changes depending on the number of groups the event affects, so use the `num_elements()` command to get the correct array size
the returned indices come before any allow or disallow group commands, and so it can be used to analyze the mapping of the instrument

%GROUPS_SELECTED[<group-idx>]

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

If a group is selected for editing the corresponding array cell 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
because of this, the note number should be a value between 0 (C) and 11 (B)

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`
The KSP timer is based on the CPU clock and thus runs at a constant rate, regardless of whether or not KONTAKT is being used in real-time. As such, it should be used to test the efficienty of script and not to make musical calculations – for musical calculations use the `$ENGINE_UPTIME` timer.

$NI_ASYNC_EXIT_STATUS

returns a value of 1 if the command that triggered the `on async_complete` callback has successfully completed its action. 0 if the command could not complete its action (e.g. file not found)

$NI_ASYNC_ID

returns the ID of the command that triggered the `on async_complete` callback.

$NI_BUS_OFFSET

to be used in the <generic> part of the engine parameter commands to point to the instrument bus level. Add the index of the bus you wish to address, for example, `$NI_BUS_OFFSET + 2` will point to instrument bus 3.

$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.

$NI_TRANSPORT_RUNNING

1 if the host's transport is running, 0 otherwise

%NOTE_DURATION[<note-number>]

note length since note-start in microseconds 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
this is not a constant and thus cannot be used to define the size of an array

$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

Specific

Callback Type Variables and Constants

$NI_CALLBACK_ID
returns the ID number of the callback. Every callback has a unique ID number which remains the same within a function.

$NI_CALLBACK_TYPE
returns the callback type. Useful for retrieving the callback which triggered a specific function.
The following constants are available:

$NI_CB_TYPE_ASYNC_OUT
$NI_CB_TYPE_CONTROLLER
$NI_CB_TYPE_INIT
$NI_CB_TYPE_LISTENER
$NI_CB_TYPE_NOTE
$NI_CB_TYPE_PERSISTENCE_CHANGED
$NI_CB_TYPE_PGS
$NI_CB_TYPE_POLY_AT
$NI_CB_TYPE_RELEASE
$NI_CB_TYPE_RPN/$NI_CB_TYPE_NRPN
$NI_CB_TYPE_UI_CONTROL
$NI_CB_TYPE_UI_UPDATE

$NI_CB_TYPE_MIDI_IN

Listener Constants

can be used with `set_listener()` or `change_listener_par()`to set which signals will trigger the `on listener` callback. Can also be used with `$NI_SIGNAL_TYPE` to determine which signal type triggered the callback.

$NI_SIGNAL_TRANSP_STOP
$NI_SIGNAL_TRANSP_START
$NI_SIGNAL_TIMER_MS
$NI_SIGNAL_TIMER_BEAT

Path Variables

$GET_FOLDER_LIBRARY_DIR
if used with an nki belonging to an encoded library: library folder
if used with an unencoded nki: the user content directory

$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.

Waveform Flag Constants	
to be used with `attach_zone()` you can combine flat constants using the bitwise `.or.`	
`$UI_WAVEFORM_USE_SLICES`	display the zone's slice markers
`$UI_WAVEFORM_USE_TABLE`	display a per slice table note: this only works if the slice markers are also active
`$UI_WAVEFORM_TABLE_IS_BIPOLAR`	make the table bipolar
`$UI_WAVEFORM_USE_MIDI_DRAG`	display a MIDI drag and drop icon note: this only works if the slice markers are also active

Waveform Property Constants	
to be used with `get/set_ui_wf_property()`	
`$UI_WF_PROP_PLAY_CURSOR`	sets or returns the play head position
`$UI_WF_PROP_FLAGS`	used to set new flag constants after the `attach_zone()` command is used
`$UI_WF_PROP_TABLE_VAL`	sets or returns the value of the indexed slice's table
`$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

General

$CONTROL_PAR_NONE

nothing will be applied to the control

$CONTROL_PAR_POS_X

sets the horizontal position in pixels

$CONTROL_PAR_POS_Y

sets the vertical position in pixels

$CONTROL_PAR_GRID_X

sets the horizontal position in grid units

$CONTROL_PAR_GRID_Y

sets the vertical position in grid units

$CONTROL_PAR_WIDTH

sets the width of the control in pixels

$CONTROL_PAR_HEIGHT

sets the height of the control in pixels

$CONTROL_PAR_GRID_WIDTH

sets the width of the control in grid units

$CONTROL_PAR_GRID_HEIGHT

sets the height of the control in grid units

$CONTROL_PAR_HIDE

sets the hide status. Can be used with the following built in 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}
$HIDE_PART_NOTHING {Show all}
$HIDE_WHOLE_CONTROL

$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. An extension is not required for the picture name, neither is the full path. If the nki references a resource container, KONTAKT will look for the file in the pictures subfolder. If the nki does not reference a resource container, it will first look in the user pictures folder (located in user/documents/Native Instruments/Kontakt 5/pictures), then in the KONTAKT pictures folder.

$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()`
this is also the value/string published to the host when using automation
also works for switches

$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
settings are relative to the size of the slider picture
negative values give a vertical slider behavior, positive values give a horizontal behaviour

$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 5 factory fonts can be used, the font itself is designated by a number (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:
0: left
1: centered
2: right

$CONTROL_PAR_SHOW_ARROWS

hides the arrows of value edits:
0: arrows are hidden
1: arrows are shown

$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 for all automatable controls. Can only be used in the init callback.

$CONTROL_PAR_KEY_SHIFT

returns 1 when the shift key was pressed (0 otherwise) while clicking the UI control.
Menus and value edits are not supported.

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

$CONTROL_PAR_KEY_ALT

returns 1 if the ALT key (PC) or OPT key (Mac) was pressed (0 otherwise) while clicking the UI control.
Menus and value edits are not supported.

$CONTROL_PAR_KEY_CONTROL

returns 1 if the CTRL key (PC) or Cmd key (Mac) was pressed (0 otherwise) while clicking the UI control.
Menus and value edits are not supported.

Specific

Tables and Waveform

$CONTROL_PAR_BAR_COLOR

sets the color of the step bar in UI tables and UI waveforms
colors are set using a hex value in the following format:
`9ff0000h {red}`
the **9** at the start is just to let KONTAKT know the value is a number, the **h** at the end is to indicate that it is a hexadecimal value.

$CONTROL_PAR_ZERO_LINE_COLOR

sets the color of the middle line in UI tables

Menus

$CONTROL_PAR_NUM_ITEMS

returns the number of menu entries of a specific dropdown menu.
Only works with `get_control_par()`.

$CONTROL_PAR_SELECTED_ITEM_IDX

returns the index of the currently selected menu entry.
Only works with `get_control_par()`.

Labels

$CONTROL_PAR_DND_BEHAVIOUR

Using a value of 1 with this variable sets the label as a "Drag and Drop" area, allowing the user to export the MIDI object currently held in the script memory by a simple drag and drop action. See the section on MIDI Object Commands for more information on MIDI handling in KSP.

UI Level Meter

$CONTROL_PAR_BG_COLOR

sets the background color of the UI level meter

colors are set using a hex value in the following format:
`9ff0000h {red}`
the **9** at the start is just to let KONTAKT know the value is a number, the **h** at the end is to indicate that it is a hexadecimal value.

$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)

File Browser

$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. Can only be used in the init callback.

$CONTROL_PAR_FILE_TYPE

sets the file type for file selector. Can only be used in the init callback.
The following file types are available:

```
$NI_FILE_TYPE_MIDI
$NI_FILE_TYPE_AUDIO
$NI_FILE_TYPE_ARRAY
```

Instrument Icon and Wallpaper

$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 and it won't be checked in the samples missing dialog when loading the wallpaper from a resource container.
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 Variables

Instrument, Source and Amp Module

$ENGINE_PAR_VOLUME
instrument/group/bus volume

$ENGINE_PAR_PAN
instrument/group/bus panorama

$ENGINE_PAR_TUNE
instrument/group/bus tuning

Source Module
$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 $ENGINE_PAR_ENVELOPE_ORDER $ENGINE_PAR_FORMANT_SHIFT

$ENGINE_PAR_OUTPUT_CHANNEL
designates the output for the group or bus 0 routes to one of KONTAKT's outputs (this bypasses the instrument insert effects) -1 routes to the instrument output (default) -2 routes to the instrument output with the instrument insert effects bypassed $NI_BUS_OFFSET + [0 – 15] routes to one of the busses (busses cannot be routed to other busses)

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
```

Jump Amp

```
$ENGINE_PAR_JMP_PREAMP
$ENGINE_PAR_JMP_BASS
$ENGINE_PAR_JMP_MID
$ENGINE_PAR_JMP_TREBLE
$ENGINE_PAR_JMP_MASTER
$ENGINE_PAR_JMP_PRESENCE
$ENGINE_PAR_JMP_HIGAIN
$ENGINE_PAR_JMP_MONO
```

Feedback Compressor

```
$ENGINE_PAR_FCOMP_INPUT
$ENGINE_PAR_FCOMP_RATIO
$ENGINE_PAR_FCOMP_ATTACK
$ENGINE_PAR_FCOMP_RELEASE
$ENGINE_PAR_FCOMP_MAKEUP
$ENGINE_PAR_FCOMP_MIX
```

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

$ENGINE_PAR_GAIN

Gain control for the KONTAKT 5 Ladder and Daft filter types.

$ENGINE_PAR_BANDWIDTH

Bandwidth control, found on the following filter types:
SV Par. LP/HP
SV Par. BP/BP
SV Ser. LP/HP

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

Formant Filters

$ENGINE_PAR_FORMANT_TALK
$ENGINE_PAR_FORMANT_SHARP
$ENGINE_PAR_FORMANT_SIZE

Simple Filter

$ENGINE_PAR_LP_CUTOFF
$ENGINE_PAR_HP_CUTOFF

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
```

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_TRANS_MASTER
$EFFECT_TYPE_BUS_COMP
$EFFECT_TYPE_TAPE_SAT
$EFFECT_TYPE_SOLID_GEQ
$EFFECT_TYPE_JUMP
$EFFECT_TYPE_FB_COMP

$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
$FILTER_TYPE_DAFT_LP
$FILTER_TYPE_SV_LP1
$FILTER_TYPE_SV_LP2
$FILTER_TYPE_SV_LP4
$FILTER_TYPE_LDR_LP1
$FILTER_TYPE_LDR_LP2
$FILTER_TYPE_LDR_LP3
$FILTER_TYPE_LDR_LP4
$FILTER_TYPE_AR_LP2
$FILTER_TYPE_AR_LP4
$FILTER_TYPE_AR_LP24
$FILTER_TYPE_SV_HP1
$FILTER_TYPE_SV_HP2
$FILTER_TYPE_SV_HP4
$FILTER_TYPE_LDR_HP1
$FILTER_TYPE_LDR_HP2
$FILTER_TYPE_LDR_HP3
$FILTER_TYPE_LDR_HP4
$FILTER_TYPE_AR_HP2
$FILTER_TYPE_AR_HP4
$FILTER_TYPE_AR_HP24
$FILTER_TYPE_DAFT_HP
$FILTER_TYPE_SV_BP2
$FILTER_TYPE_SV_BP4
$FILTER_TYPE_LDR_BP2
$FILTER_TYPE_LDR_BP4
$FILTER_TYPE_AR_BP2
$FILTER_TYPE_AR_BP4
$FILTER_TYPE_AR_BP24
$FILTER_TYPE_SV_NOTCH4
$FILTER_TYPE_LDR_PEAK
$FILTER_TYPE_LDR_NOTCH
$FILTER_TYPE_SV_PAR_LPHP
$FILTER_TYPE_SV_PAR_BPBP
$FILTER_TYPE_SV_SER_LPHP
$FILTER_TYPE_FORMANT_1
$FILTER_TYPE_FORMANT_2
$FILTER_TYPE_SIMPLE_LPHP
Note that the Solid G-EQ is not treated as a filter/EQ subtype, but as an effect.

$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 possible to send and receive values from one script to another, discarding the usual left-to-right order by using the 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 callback 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.

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

Resource Container

Introduction

The Resource Container is a useful tool for library developers. It is a dedicated location to store scripts, graphics, .nka files and impulse response files that can be referenced by any NKI or group of NKIs that are linked to the container. Another benefit is that you can create a 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. When loading an NKI, the resource container is treated like a sample, so if it is not found it will appear in the Samples Missing dialogue.

Setup

To create a Resource Container for your NKI, open up its instrument options and click the <Create> button beside the area labeled as Resource Container. After creating a new resource container file, KONTAKT checks if there is already a resource folder structure available. If there isn't, you can let KONTAKT create it for you. If you do this, you will find a Resources and a Data folder next to the NKR file you just created.
The Resources folder is the place where you can store the files an NKI can use that are not samples. As you can see KONTAKT has already created several subfolders for you: ir_samples, pictures (for GUI graphics and wallpapers), data (for .nka files) 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 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... -> Resources 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.
- check the data subfolder for any .nka files which will then be put into the monolith.

After that rename your Resources folder and reopen your NKI. Now that there is no Resources folder present anymore, KONTAKT will automatically read from 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 work 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.

Multi Script

General Information

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 `on 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.

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

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
```

Version History

KONTAKT 5.4.2

Improved Features

- various manual corrections

KONTAKT 5.4.1

New Features

- New callback type: `on persistence_changed`
- New command: `set_snapshot_type()`
- New command: `make_instr_persistence()`
- New key color constants and command: `get_key_color()`
- Ability to set the pressed state of KONTAKT's keyboard: `set_key_pressed()`, `set_key_pressed_support(), get_key_triggerstate()`
- Ability to specify key names and ranges: `set_key_name()`, `get_key_name()`, `set_keyrange()`, `remove_keyrange()`
- Ability to specify key types: `set_key_type()`, `get_key_type()`

Improved Features

- Data folder in ressource container, additional mode for `load_array()`
- Usage of `load_array_str()`in other callbacks

KONTAKT 5.3

New Features

- Added Engine Parameter Variables for the new Simple Filter effect

KONTAKT 5.2

Improved Features

- Updated MIDI file handling

New Features

- Commands to insert and remove MIDI events

KONTAKT 5.1.1

New Features

- Added Engine Parameter Variables for the new Feedback Compressor effect

KONTAKT 5.1

New Features

- new commands: `load_array_str()`, `save_array_str()`
- Added Engine Parameter Variables for the new Jump Amp effect

Manual Corrections

- miscellaneous corrections and improvements

KONTAKT 5.0.2

New Features

- New Engine Parameter Variables for Time Machine Pro (HQ Mode): `$ENGINE_PAR_ENVELOPE_ORDER`, `$ENGINE_PAR_FORMANT_SHIFT`

KONTAKT 5.0.1

New Features

- Added effect type and effect sub-type constants for the new KONTAKT 5 effects

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`
  incl. new commands and built-in variables: `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`
  incl. new commands and built-in variables: `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`
  incl. new built-in variables: `$NI_ASYNC_ID, $NI_ASYNC_EXIT_STATUS, $NI_CB_TYPE_ASYNC_OUT`
- new internal constant for KONTAKT's new bus system: `$NI_BUS_OFFSET`
- new engine_par constants for new KONTAKT 5 effects
- new commands: `wait_ticks(), stop_wait()`

Improved Features

- support for string arrays added for `load array()` and `save_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 for creating instrument libraries
- new ID to set wallpapers via script: `$INST_WALLPAPER_ID`
- new key color: `$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()`
- 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 affecting 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

I

L

M

N

O

P

R

S

T

U

W