



KSP Reference Manual

Table of Contents

1. Disclaimer	1
2. Welcome to KSP	2
3. Callbacks	3
General Information	3
on async_complete	4
on controller	6
on init	7
on listener	9
on note	10
on persistence_changed	11
on pgs_changed	12
on poly_at	13
on release	14
on rpn/nrpn	15
on ui_control()	16
on ui_update	17
4. Variables	18
General Information	18
\$ (int variable)	19
% (int array)	20
~ (real variable)	21
? (real array)	22
@ (string variable)	23
! (string array)	24
const \$ (constant integer)	25
const ~ (real constant)	26
polyphonic \$ (polyphonic integer)	27
make_instr_persistent()	28
make_persistent()	29
read_persistent_var()	30
watch_var()	31
watch_array_idx()	32
5. Control Statements	33
if...else...end	33
select()	34
while()	35
Boolean Operators	36
6. User Interface Controls	37
ui_button	37
ui_file_selector	38
ui_label	40
ui_knob	41
ui_level_meter	42
ui_menu	43
ui_mouse_area	44

ui_panel	45
ui_slider	46
ui_switch	47
ui_table	48
ui_text_edit	49
ui_value_edit	50
ui_waveform	51
ui_wavetable	52
ui_xy	53
7. Arithmetic Commands & Operators	55
Basic Operators	55
Integer Operators & Commands	56
Real Number Commands	57
Rounding Commands	58
Trigonometric Commands	59
Bit Operators	60
random()	61
int_to_real()	62
real_to_int()	63
msb()	64
lsb()	65
8. General Commands	66
disable_logging()	66
exit	67
ignore_controller	68
message()	69
note_off()	70
play_note()	71
set_controller()	72
set_rpn()/set_nrpn()	73
set_snapshot_type()	74
9. Event Commands	75
by_marks()	75
change_note()	76
change_pan()	77
change_tune()	78
change_velo()	79
change_vol()	80
delete_event_mark()	81
event_status()	82
fade_in()	83
fade_out()	84
get_event_ids()	85
get_event_mark()	86
get_event_par()	87
get_event_par_arr()	89
ignore_event()	90
redirect_output()	91
set_event_mark()	92
set_event_par()	93

set_event_par_arr()	95
10. Array Commands	97
array_equal()	97
num_elements()	98
search()	99
sort()	100
11. Group Commands	101
allow_group()	101
disallow_group()	102
find_group()	103
get_purge_state()	104
group_name()	105
purge_group()	106
12. Time-Related Commands	107
change_listener_par()	107
ms_to_ticks()	108
set_listener()	109
stop_wait()	110
reset_ksp_timer	111
ticks_to_ms()	112
wait()	113
wait_async()	114
wait_ticks()	115
13. User Interface Commands	116
add_menu_item()	116
add_text_line()	117
attach_level_meter()	118
attach_zone()	120
fs_get_filename()	121
fs_navigate()	122
get_control_par()	123
get_font_id()	124
get_menu_item_str()	125
get_menu_item_value()	126
get_menu_item_visibility()	127
get_ui_id()	128
get_ui_wf_property()	129
hide_part()	130
load_performance_view()	131
make_perfview	132
move_control()	133
move_control_px()	134
set_control_help()	135
set_control_par()	136
set_control_par_arr()	137
set_knob_defval()	138
set_knob_label()	139
set_knob_unit()	140
set_menu_item_str()	141

set_menu_item_value()	142
set_menu_item_visibility()	143
set_table_steps_shown()	144
set_script_title()	145
set_skin_offset()	146
set_text()	147
set_ui_color()	148
set_ui_height()	149
set_ui_height_px()	150
set_ui_width_px()	151
set_ui_wf_property()	152
14. Keyboard Commands	153
get_key_color()	153
get_key_name()	154
get_key_triggerstate()	155
get_key_type()	156
get_keyrange_min_note()	157
get_keyrange_max_note()	158
get_keyrange_name()	159
set_key_color()	160
set_key_name()	162
set_key_pressed()	163
set_key_pressed_support()	164
set_key_type()	165
set_keyrange()	166
remove_keyrange()	167
15. Engine Parameter Commands	168
find_mod()	168
find_target()	170
get_engine_par()	171
get_engine_par_disp()	173
get_voice_limit()	174
output_channel_name()	175
set_engine_par()	176
set_voice_limit()	178
16. Zone Commands	179
General Information	179
get_loop_par()	180
get_sample()	181
get_zone_par()	182
is_zone_empty()	183
set_loop_par()	184
set_num_user_zones()	185
set_sample	186
set_zone_par()	187
17. Load/Save Commands	188
General Information	188
get_folder()	189
load_array()	190

load_array_str()	192
load_ir_sample()	194
save_array()	196
save_array_str()	197
save_midi_file()	199
18. Music Information Retrieval	200
General Information	200
detect_pitch()	201
detect_loudness()	202
detect_peak()	203
detect_rms()	204
detect_sample_type()	205
detect_drum_type()	206
detect_instrument_type()	207
Examples	208
19. MIDI Object Commands	209
General Information	209
mf_insert_file()	210
mf_set_export_area()	212
mf_set_num_export_areas()	213
mf_copy_export_area()	214
mf_set_buffer_size()	215
mf_get_buffer_size()	216
mf_reset()	217
mf_insert_event()	218
mf_remove_event()	219
mf_set_event_par()	220
mf_get_event_par()	221
mf_get_id()	222
mf_set_mark()	223
mf_get_mark()	224
by_marks()	225
by_track()	226
mf_get_first()	227
mf_get_last()	228
mf_get_next()	229
mf_get_next_at()	230
mf_get_prev()	231
mf_get_prev_at()	232
mf_get_num_tracks()	233
20. Built-in Variables and Constants	234
General	234
Events and MIDI	236
Transport and Timing	239
Callbacks and UI	241
Mathematical Constants	243
21. Control Parameters	244
General	244
Specific	249

22. Engine Parameters	258
Instrument, Source and Amp Module	258
Insert Effects	261
Filter and EQ	270
Send Effects	272
Modulation	275
Module Types and Subtypes	277
Group Start Options Query	281
23. Zone Parameters	282
Zone Parameters	282
Loop Parameters	284
Sample Parameters	285
24. Advanced Concepts	286
Preprocessor & System Scripts	286
PGS	289
Zone and Slice Functions	291
User-defined Functions	292
Resource Container	293
Changing FX from KSP	295
The Advanced Engine Tab	297
25. Multi Script	298
General Information	298
ignore_midi	299
on midi_in	300
set_midi()	301
Multi Script Command Arguments	302
26. New Features	304
KONTAKT 6.7.0	304
KONTAKT 6.6.0	305
KONTAKT 6.5.0	306
KONTAKT 6.4.0	307
KONTAKT 6.3.0	308
KONTAKT 6.2.0	309
KONTAKT 6.1.0	310
KONTAKT 6.0.2	311
KONTAKT 5.8.0	312
KONTAKT 5.7.0	313
KONTAKT 5.6.8	314
KONTAKT 5.6.5	315
KONTAKT 5.6.0	316
KONTAKT 5.5.0	317
KONTAKT 5.4.2	318
KONTAKT 5.4.1	319
KONTAKT 5.3.0	320
KONTAKT 5.2.0	321
KONTAKT 5.1.1	322
KONTAKT 5.1.0	323
KONTAKT 5.0.2	324
KONTAKT 5.0.1	325

KONTAKT 5	326
KONTAKT 4.2.0	327
KONTAKT 4.1.2	328
KONTAKT 4.1.1	329
KONTAKT 4.0.2	330
KONTAKT 4.1.0	331
KONTAKT 4	332
KONTAKT 3.5.0	333
KONTAKT 3	334
KONTAKT 2.2.0	335
KONTAKT 2.1.1	336
KONTAKT 2.1.0	337
KONTAKT 2	338

1. Disclaimer

The information in this document is subject to change without notice and does not represent a commitment on the part of Native Instruments GmbH. The software described by this document is subject to a License Agreement and may not be copied to other media. No part of this publication may be copied, reproduced or otherwise transmitted or recorded, for any purpose, without prior written permission by Native Instruments GmbH, hereinafter referred to as Native Instruments.

“Native Instruments”, “NI” and associated logos are (registered) trademarks of Native Instruments GmbH.

Mac, macOS, GarageBand, Logic and iTunes are registered trademarks of Apple Inc., registered in the U.S. and other countries.

All other trademarks are the property of their respective owners and use of them does not imply any affiliation with or endorsement by them.

Document authored by: Nikolas Jeroma, Adam Hanley, Mario Krušelj, Elpiniki Pappa, Dinos Vallianatos, Yaron Eshkar, Hannah Lockwood

Software version: 6.7.0 (2/2022)

2. Welcome to KSP

Welcome to the KONTAKT Script Processor (KSP) reference manual. KSP is the technology that powers thousands of instruments in our industry leading sampling platform KONTAKT. Its underlying KONTAKT scripting language defines how samples are being played, how MIDI information is used for interacting with an instrument, how the instrument looks and much more. From simple sequencers to large orchestral clusters with thousands of samples, KSP makes it work.

This resource is a reference manual that covers every function, command, variable, callback and other element of the KONTAKT scripting language. Where applicable, it also includes examples of short code snippets that demonstrate how a given function can be used. To learn more about KONTAKT, refer to the [KONTAKT User Manual](#).

We hope you enjoy exploring KSP!

3. 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 **exit** command.
- 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](#)

[stop_wait\(\)](#)

on async_complete

on async_complete

async complete callback, triggered after the execution of any load/save command or other commands which are async-activated.

Remarks

- To resolve synchronization issues, the commands listed in the "See Also" section below 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 file is loaded

See Also

[Load/Save Commands](#)

[set_voice_limit\(\)](#)

[save_midi_file\(\)](#)

[mf_insert_file\(\)](#)

[mf_set_buffer_size\(\)](#)

[mf_reset\(\)](#)

[set_engine_par\(\)](#)

[set_zone_par\(\)](#)

[set_loop_par\(\)](#)

[set_sample](#)

[purge_group\(\)](#)

`load_ir_sample()`

Music Information Retrieval: MIR Commands

Built-in Variables and Constants: `$NI_ASYNC_EXIT_STATUS`, `$NI_ASYNC_ID`

Module Types and Subtypes: `$ENGINE_PAR_EFFECT_TYPE`, `$ENGINE_PAR_EFFECT_SUBTYPE`

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](#)

Events and MIDI: %CC[], \$CC_NUM, \$VCC_PITCH_BEND, \$VCC_MONO_AT

on init

on init

Initialization callback, executed when the script was successfully compiled without warnings or errors.

Remarks

The init callback will be executed when:

- Clicking the "Apply" button
- 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\(\)](#)

[Callbacks and UI: \\$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\(\)](#)

[Events and MIDI: \\$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 instrument has been loaded. This 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
```

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: `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

[Events and MIDI](#): %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\(\)](#)

[get_event_par\(\)](#): \$EVENT_PAR_REL_VELOCITY

on rpn/nrpn

on rpn/nrpn

RPN and NRPN callbacks, executed whenever a RPN or NRPN (registered/non-registered 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\(\)](#)

[Events and MIDI](#): \$RPN_ADDRESS, \$RPN_VALUE

on ui_control()

on ui_control(<variable>)

UI callback, executed whenever the user interacts with 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\(\)](#)

4. 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 (`_`).
- Please do not create variables with the prefixes below, as these prefixes are used for internal variables and constants.

`$NI_`

`$CONTROL_PAR_`

`$EVENT_PAR_`

`$ENGINE_PAR_`

\$ (int variable)

```
declare $<int variable>
```

Declare a user-defined 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, similarly as above but with in-line value initialization

See Also

[on init](#)

[make_persistent\(\)](#)

[read_persistent_var\(\)](#)

[int_to_real\(\)](#)

[real_to_int\(\)](#)

% (int array)

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

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

Remarks

- The maximum size of arrays is 1000000 indices.
- 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, refer to 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 a specific value

See Also

[Array Commands](#)

[Group Commands](#)

[make_persistent\(\)](#)

~ (real variable)

```
declare ~<real variable>
```

Declare a user-defined variable to store a single real value

Remarks

- Real numbers must always be defined with a decimal point, even if the number is a whole number. For example 2.0 should be used instead of only 2.

Examples

```
on init
  declare ~test
  ~test := 0.5
end on
```

Creating a variable

```
on init
  declare ~test := 0.5
end on
```

Creating a variable, the same as above but shorter

See Also

[on init](#)

[make_persistent\(\)](#)

[read_persistent_var\(\)](#)

[int_to_real\(\)](#)

[real_to_int\(\)](#)

? (real array)

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

Declare a user-defined array to store single real values at specific indices

Remarks

- The maximum size of arrays is 1000000 indices.
- The number of elements must be defined with a constant real value, a standard variable cannot be used.
- It is possible to initialize an array with one value, refer to the second example below.
- The commands `array_equal()` and `search()` do not work with arrays of real numbers.

Examples

```
on init
  declare ?presets[5*4] := (...
    {1}    1.0, 1.0, 1.0, 1.0, ...
    {2}    0.5, 0.7, 0.1, 0.5, ...
    {3}    1.0, 0.6, 0.6, 0.2, ...
    {4}    0.0, 0.0, 0.5, 0.3, ...
    {5}    0.0, 1.0, 0.4, 0.1)
end on
```

Creating an array for storing preset data

```
on init
  declare ?presets[10*8] := (1.0)
end on
```

Quick way of initializing the same array with a specific value

See Also

[Array Commands](#)

[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.
- The maximum length of text that can be stored in a string variable is 320 characters.

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

- The maximum size of arrays is 1000000 indices.
- Just like with string variables, the contents of a string array cannot be defined on the same line as the declaration.
- The maximum length of a string at any given indice is 320 characters.

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

[@ \(string variable\)](#)

const \$ (constant integer)

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

Declare a user-defined constant 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](#)

const ~ (real constant)

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

Declare a user-defined constant to store a single real 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 ~BIG_NUMBER := 100000.0
  declare const ~SMALL_NUMBER := 0.00001
end on
```

See Also

[on init](#)

polyphonic \$ (polyphonic integer)

```
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 can 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

make_instr_persistent()

make_instr_persistent(<variable>)

Retain the value of a variable within the instrument only

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 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 with 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 updating script code by replacing old code with new one, the values of persistent variables will be retained.
- Sometimes, when working on more complex scripts, you might want to flush the values of persistent variables by resetting the script. You can do this by loading an empty script slot from the Script Editor preset menu, then applying your code again.

Examples

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

User interface elements, such as knobs, should usually retain their value when reloading the instrument

See Also

[read_persistent_var\(\)](#)

[on persistence_changed](#)

[make_instr_persistent\(\)](#)

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 updating script code by replacing old code with new one, the values of persistent variables will be retained.
- Sometimes, when working on more complex scripts, you might want to flush the values of persistent variables by resetting the script. You can do this by loading an empty script slot from the Script Editor preset menu, then applying your code again.
- 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.

See Also

[make_persistent\(\)](#)

[on persistence_changed](#)

watch_var()

watch_var(<variable>)

sends an event to the Creator Tools KSP Log for every change of the watched variable's value

Remarks

- This command can only be used within the init callback.
- This command has no effect on KONTAKT's status bar – the events only appear in Creator Tools.
- This command does not work with built-in variables

Examples

```
on init
  declare $intVar
  watch_var($intVar)
  make_persistent($intVar)

end on

on note
  $intVar := $EVENT_VELOCITY
end on
```

Try playing some notes while having Creator Tools running. Make sure you have the Variable Watching panel of the Debugger tool open.

watch_array_idx()

watch_array_idx(<array>, <array_idx>)

sends an event to the Creator Tools KSP Log for every change of the watched array cell's value

Remarks

- This command can only be used within the init callback.
- This command has no effect on KONTAKT's status bar – the events only appear in Creator Tools.
- This command does not work with built-in arrays

Examples

```
on init
  declare %mykeys[128]
  watch_array_idx(%mykeys,60)
  watch_array_idx(%mykeys,61)
  watch_array_idx(%mykeys,62)
  watch_array_idx(%mykeys,63)
  watch_array_idx(%mykeys,64)

  declare ui_button
$save
  declare ui_button
$load
end on

on note
  %mykeys[$EVENT_NOTE]
:= $EVENT_VELOCITY
end on

on ui_control($save)
  save_array(%mykeys,0)
end on

on ui_control($load)
  load_array(%mykeys,0)
end on
```

Try playing some notes or clicking on the save and load buttons while having Creator Tools running. Make sure you have the Variable Watching panel of the Debugger tool open.

5. Control Statements

if...else...end

if...else...end if

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](#)

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

[wait\(\)](#)

[Events and MIDI](#): \$NOTE_HELD

Boolean Operators

Boolean Operators	
<code>x > y</code>	Greater than
<code>x < y</code>	Less than
<code>x >= y</code>	Greater than or equal
<code>x <= y</code>	Less than or equal
<code>x = y</code>	Equal
<code>x # y</code>	Not equal
<code>in_range(x,y,z)</code>	True if x is between y and z
<code>not a</code>	True if a is false and vice versa
<code>a and b</code>	True if a is true and b is true
<code>a or b</code>	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.

6. 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 (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_file_selector

declare ui_file_selector \$<variable-name>

Create a file selector

Examples

```
on init
    message("")
    set_ui_height(5)

    {set browser path here, for example:
    @basepath := "/Users/username/Desktop/MIDI Files/"}
    declare @basepath
    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 := -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 ($prev)
    {calls 'on ui_control ($file_browser)'}
    fs_navigate($browser_id,0)
```

```
        $prev := 0
end on

on ui_control ($next)
    {calls 'on ui_control ($file_browser)'}
    fs_navigate($browser_id,1)
    $next := 0
end on
```

Loading MIDI files via UI file selector

See Also

[fs_navigate\(\)](#)

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 sizes

```
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_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_level_meter

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

Create a level meter

Remarks

- The level meter can display the output levels of buses and the main instrument output. It can also display the gain reduction from compressor and limiter effects.

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 displaying one channel of KONTAKT's instrument output

See Also

[attach_level_meter\(\)](#)

Specific: \$CONTROL_PAR_BG_COLOR, \$CONTROL_PAR_OFF_COLOR,
\$CONTROL_PAR_ON_COLOR, \$CONTROL_PAR_OVERLOAD_COLOR,
\$CONTROL_PAR_PEAK_COLOR, \$CONTROL_PAR_VERTICAL, \$CONTROL_PAR_RANGE_MIN,
\$CONTROL_PAR_RANGE_MAX

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
```

Quickly create a menu with many entries

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_mouse_area

declare ui_mouse_area

Create a user interface mouse area.

Remarks

- A mouse area supports drag and drop of the following file types: audio (WAV, AIF, AIFF, NCW), MIDI and array (NKA).
- It is possible to define which types of files are accepted as drop targets, and whether to accept just one or multiple files.
- The mouse area widget is invisible, but the drop target can be shown or hidden like any other UI widget.

Examples

```
on init
  declare ui_mouse_area $waveDnD
  set_control_par(get_ui_id($waveDnD),
    $CONTROL_PAR_DND_ACCEPT_AUDIO, $NI_DND_ACCEPT_ONE)
  set_control_par(get_ui_id($waveDnD),
    $CONTROL_PAR_DND_ACCEPT_ARRAY,
    $NI_DND_ACCEPT_ONE)set_control_par(get_ui_id($waveDnD),
    $CONTROL_PAR_WIDTH,90)set_control_par(get_ui_id($waveDnD),
    $CONTROL_PAR_HEIGHT,32)set_control_par(get_ui_id($waveDnD),
    $CONTROL_PAR_RECEIVE_DRAG_EVENTS, 1)
end on
```

The on ui_control callback is triggered by a drop action. It has 3 built-in arrays:

```
!NI_DND_ITEMS_AUDIO
!NI_DND_ITEMS_MIDI
!NI_DND_ITEMS_ARRAY
```

Example UI callback

```
on ui_control ($waveDnD)
  if ($NI_MOUSE_EVENT_TYPE = $NI_MOUSE_EVENT_TYPE_DRAG)
    message("DRAG")
    message("MOUSE OVER CONTROL: " & $NI_MOUSE_OVER_CONTROL)
  end if

  if ($NI_MOUSE_EVENT_TYPE = $NI_MOUSE_EVENT_TYPE_DROP)
    if (num_elements(!NI_DND_ITEMS_AUDIO) = 1)
      wait_async(set_sample(%NI_USER_ZONE_IDS[0],
        !NI_DND_ITEMS_AUDIO[0]))
    end if
  end if
end on
```

See Also

Specific: `$NI_MOUSE_EVENT_TYPE`, `$NI_MOUSE_EVENT_TYPE_DND_DROP`,
`$NI_MOUSE_EVENT_TYPE_DND_DRAG`, `$NI_MOUSE_OVER_CONTROL`

ui_panel

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

Create a user interface panel

Remarks

A panel is a control that can contain one or multiple controls. Unlike the rest of the UI control types, panels don't have size. They are very useful for grouping controls that are meant to be handled together, then one can simultaneously modify the **\$CONTROL_PAR_HIDE**, **\$CONTROL_PAR_POS_X**, **\$CONTROL_PAR_POS_Y** or **\$CONTROL_PAR_Z_LAYER** properties of all the controls contained in that panel. The position of a contained control is relative to the panel's position. This means that the control's (0,0) position is the current (x,y) position of the panel.

Panels can be nested, so they can contain other panels. If panelA is contained in panelB, then panelA will appear in front of panelB. This is because children panels have a higher Z-layer value than their parent panels. One could use this logic to easily create hierarchies in a performance view.

Examples

```
on init
    declare ui_panel $mixer
    declare ui_knob $volume(0,300,1)

    set_control_par(get_ui_id($volume), $CONTROL_PAR_PARENT_PANEL, get_ui_id($mixer))
end on
```

Adds the volume knob in the mixer panel

See Also

General: [\\$CONTROL_PAR_PARENT_PANEL](#)

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_arr\(\)](#)

Specific: [\\$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 (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.

Remarks

- The maximum number of columns in a ui_table is 128.

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

```
on init
    declare ui_table %table[20] (4,4,100)
    declare ui_button $button
end on

on ui_control($button)
    if($button = 1)
        hide_part(%table,$HIDE_PART_VALUE)
    else
        hide_part(%table,$HIDE_PART_NOTHING)
    end if
end on
```

Hiding a table value

See Also

[set_table_steps_shown\(\)](#)

[hide_part\(\)](#)

Specific: \$NI_CONTROL_PAR_IDX

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

Specific: [\\$CONTROL_PAR_SHOW_ARROWS](#), [\\$VALUE_EDIT_MODE_NOTE_NAMES](#)

ui_waveform

declare ui_waveform \$<variable>(<width>, <height>)

Create a waveform control to display zones and slices. This can also be used to control specific parameters per slice and for drag and 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. Use a sample named Test to test the above code.

See Also

[set_ui_wf_property\(\)](#)

[get_ui_wf_property\(\)](#)

[attach_zone\(\)](#)

[Zone and Slice Functions: find_zone\(\)](#)

Specific: Waveform Flag Constants, Waveform Property

Constants, \$CONTROL_PAR_WAVE_COLOR, \$CONTROL_PAR_BG_COLOR,

\$CONTROL_PAR_WAVE_CURSOR_COLOR, \$CONTROL_PAR_SLICEMARKERS_COLOR,

\$CONTROL_PAR_BG_ALPHA

ui_wavetable

```
declare ui_wavetable $ <variable>
```

Create a wavetable widget, visualizing the state of a zone that is running in wavetable mode.

Examples

```
on init
    declare ui_wavetable $wavetable
    set_control_par(get_ui_id($wavetable), $CONTROL_PAR_WT_ZONE,...
                    find_zone("Wavetable01"))
end on
```

Displays the zone "Wavetable01" within the wavetable control. Use a wavetable named Wavetable01 to test the above code.

See Also

[set_control_par\(\)](#)

[Zone and Slice Functions: find_zone\(\)](#)

Specific: \$CONTROL_PAR_WT_VIS_MODE, \$NI_WT_VIS_2D,
\$NI_WT_VIS_3D, \$CONTROL_PAR_WAVE_COLOR, \$CONTROL_PAR_BG_COLOR,
\$CONTROL_PAR_BG_ALPHA, \$CONTROL_PAR_WAVE_COLOR, \$CONTROL_PAR_WAVE_ALPHA,
\$CONTROL_PAR_WAVE_END_COLOR, \$CONTROL_PAR_WAVE_END_ALPHA,
\$CONTROL_PAR_WAVETABLE_END_COLOR, \$CONTROL_PAR_WAVETABLE_END_ALPHA,
\$CONTROL_PAR_PARALLAX_X, \$CONTROL_PAR_PARALLAX_Y, \$CONTROL_PAR_WT_ZONE

ui_xy

declare ui_xy ?<array>[num-of-elements]

Create an XY pad

Remarks

- The range of each axis on the XY pad is always between 0.0 and 1.0.
- The number of cursors in the XY pad, i.e. the interactive elements, is defined by the size of the array. Each index in the array represents one axis of one cursor, so two indices are needed for each cursor. Applying this, if you wanted to create an XY pad with 3 cursors, then the size of the XY array would be 6 elements.
- The maximum size of the XY array is 32 elements, so the maximum number of cursors in the XY pad is 16.
- The even indices of the array hold the X axis value of the cursors, and the odd indices hold the Y axis values. So index 0 is the X value of the first cursor, and index 1 is the Y value of the first cursor.
- It is possible to define how the XY pad reacts to mouse interaction using the `$CONTROL_PAR_MOUSE_MODE` parameter.
- Querying `$NI_MOUSE_EVENT_TYPE` within the on `ui_control()` callback allows identification of the mouse event type that triggered it.

Examples

```
on init

    {basic initialization}
    message("")
    make_perfview

    set_ui_color(9dddddhh)
    set_ui_height_px(350)

    {create an XY pad with 2 cursors}
    declare ui_xy ?myXY[4]

    {store the UI ID of the XY pad}
    declare $xyID
    $xyID := get_ui_id(?myXY)

    {skinning the cursors}
    set_control_par_str_arr($xyID, $CONTROL_PAR_CURSOR_PICTURE, ...
        "Picture1", 0)
    set_control_par_str_arr($xyID, $CONTROL_PAR_CURSOR_PICTURE, ...
        "Picture2", 2)

    {set automation IDs and names}
    set_control_par_arr($xyID, $CONTROL_PAR_AUTOMATION_ID, 0, 0)
    set_control_par_arr($xyID, $CONTROL_PAR_AUTOMATION_ID, 1, 1)
    set_control_par_arr($xyID, $CONTROL_PAR_AUTOMATION_ID, 2, 2)
    set_control_par_arr($xyID, $CONTROL_PAR_AUTOMATION_ID, 3, 3)

    set_control_par_str_arr($xyID, $CONTROL_PAR_AUTOMATION_NAME, ...
        "Cutoff", 0)
    set_control_par_str_arr($xyID, $CONTROL_PAR_AUTOMATION_NAME, ...
        "Resonance", 1)
```

```

set_control_par_str_arr($xyID, $CONTROL_PAR_AUTOMATION_NAME, ...
    "Delay Pan", 2)
set_control_par_str_arr($xyID, $CONTROL_PAR_AUTOMATION_NAME, ...
    "Delay Feedback", 3)

{define the mouse behaviour}
set_control_par($xyID, $CONTROL_PAR_MOUSE_MODE, 0)
set_control_par($xyID, $CONTROL_PAR_MOUSE_BEHAVIOUR_X, 1000)
set_control_par($xyID, $CONTROL_PAR_MOUSE_BEHAVIOUR_Y, 1000)

{position and size}
set_control_par($xyID, $CONTROL_PAR_POS_X, 50)
set_control_par($xyID, $CONTROL_PAR_POS_Y, 50)
set_control_par($xyID, $CONTROL_PAR_WIDTH, 200)
set_control_par($xyID, $CONTROL_PAR_HEIGHT, 200)

{move the cursors to the center of the XY pad}
?myXY[0] := 0.5 {1st cursor, X axis}
?myXY[1] := 0.5 {1st cursor, Y axis}
?myXY[2] := 0.5 {2nd cursor, X axis}
?myXY[3] := 0.5 {2nd cursor, Y axis}

```

end on

Creating an XY pad control with two cursors, custom cursor images, and automation information

See Also

[set_control_par_arr\(\)](#)

General: [set_control_par_str_arr\(\)](#)

Specific: [\\$HIDE_PART_CURSOR](#), [\\$NI_CONTROL_PAR_IDX](#), [\\$CONTROL_PAR_MOUSE_MODE](#),
[\\$CONTROL_PAR_ACTIVE_INDEX](#), [\\$CONTROL_PAR_CURSOR_PICTURE](#),
[\\$CONTROL_PAR_MOUSE_BEHAVIOUR_X](#), [\\$CONTROL_PAR_MOUSE_BEHAVIOUR_Y](#)

7. Arithmetic Commands & Operators

Basic Operators

Basic operators	
The following operators work on both integers and real numbers.	
$x := y$	Assignment (the value of y is assigned to x)
$x + y$	Addition
$x - y$	Subtraction
$x * y$	Multiplication
x / y	Division
$-x$	Negative value
$\text{abs}(x)$	Absolute value

Integer Operators & Commands

The following commands and operators can only be performed on integer variables and values.

inc(x)

Increment an expression by 1 ($x + 1$)

dec(x)

Decrement an expression by 1 ($x - 1$)

x mod y

Modulo; returns the remainder of a division

e.g. $13 \bmod 8$ returns the value 5

Real Number Commands

The following commands can only be performed on real numbers.

exp(x)

Exponential function (returns the value of e^x)

log(x)

Logarithmic function

pow(x, y)

Power (returns the value of x^y)

sqrt(x)

Square root

Rounding Commands

Rounding commands can only be performed on real numbers.

ceil(x)

Ceiling (round up)

$\text{ceil}(2.3) = 3.0$

floor(x)

Floor (round down)

$\text{floor}(2.8) = 2.0$

round(x)

Round (round to nearest)

$\text{round}(2.3) = 2.0$

$\text{round}(2.8) = 3.0$

Trigonometric Commands

Trigonometric commands can only be performed on real numbers.

cos(x)

cosine function

sin(x)

sine function

tan(x)

tangent function

acos(x)

arccosine (inverse cosine function)

asin(x)

arcsine (inverse sine function)

atan(x)

arctangent (inverse tangent function)

Bit Operators

The following bit operators can be used:

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

random()

random(<min>, <max>)

Generate a random integer between (and including) <min> and <max>.

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 by ± 10 percent

int_to_real()

int_to_real(<integer value>)

Converts an integer value into a real number

Examples

```
on init
  declare ~velocity_disp
end on

on note
  ~velocity_disp := int_to_real($EVENT_VELOCITY)/127.0
  message(~velocity_disp)
end on
```

Displays the event velocity in the range from 0.0 to 1.0

See Also

[real_to_int\(\)](#)

real_to_int()

real_to_int(<real value>)

Converts a real number into an integer

Remarks

- Using this command without any rounding function will cause the real value to be truncated, so performing this function both 2.2 and 2.8 will return an integer value of 2

Examples

```
on init
  declare $test_int
  declare ~test_real := 2.8

  $test_int := real_to_int(~test_real)
  message($test_int)
end on
```

Converting a variable from real to integer and then displaying it

See Also

[int_to_real\(\)](#)

Rounding Commands: [ceil\(\)](#), [floor\(\)](#), [round\(\)](#)

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\(\)](#)

[Events and MIDI](#): \$RPN_ADDRESS, \$RPN_VALUE

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

```
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\(\)](#)

[Events and MIDI](#): \$RPN_ADDRESS, \$RPN_VALUE

8. General Commands

disable_logging()

disable_logging(<type>)

Disables emission of messages, warnings or watched variable events to both the KONTAKT status bar and Creator Tools

<type> The type of event the emission of which is deactivated. Use one of the following: `$NI_LOG_MESSAGE`, `$NI_LOG_WARNING` or `$NI_LOG_WATCHING`.

Remarks

- Only supported in the init callback

Examples

```
on init
  disable_logging($NI_LOG_MESSAGE)
  disable_logging($NI_LOG_WARNING)
  disable_logging($NI_LOG_WATCHING)
end on
```

Keep the lines above commented out while development and bring them back in right before shipping your product to disable any debugging-related content

See Also

[watch_var\(\)](#)

[watch_array_idx\(\)](#)

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 within 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

[reset_ksp_timer](#)

[ui_label](#)

[set_text\(\)](#)

Transport and Timing: [\\$ENGINE_UPTIME](#), [\\$KSP_TIMER](#)

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 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 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, pitch bend or channel pressure value

<controller>	This parameter sets the type, and in the case of MIDI CCs, sets the CC number: • A number from 0 to 127 designates a MIDI CC number • \$VCC_PITCH_BEND indicates pitch bend • \$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 • Pitch bend values go from -8192 to 8191

Remarks

- **set_controller()** cannot be used within an init callback. If for some reason you want to send a controller value upon instrument load, use **persistence_changed** callback.

```
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
```

If you have a keyboard with no aftertouch, press C1 instead

See Also

[ignore_controller](#)

Events and MIDI: **\$VCC_PITCH_BEND**, **\$VCC_MONO_AT**

set_rpn()/set_nrpn()

set_rpn(<address>, <value>)

Send a RPN or NRPN message

<address>	The RPN or NRPN address (0 - 16383)
<value>	The value of the RPN or NRPN message (0 - 16383)

Remarks

- KONTAKT cannot handle RPN or NRPN messages as external modulation sources. You can however use these message for simple inter-script communication.

See Also

[on rpn/nrpn](#)

[set_controller\(\)](#)

[msb\(\)](#)

[lsb\(\)](#)

[Events and MIDI](#): \$RPN_ADDRESS, \$RPN_VALUE

set_snapshot_type()

set_snapshot_type(<type>)

Configures the KSP processor behavior of all five slots when a snapshot is saved and 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 2: same as type 0, but only KSP variables are saved with the snapshot 3: same as type 1, but only KSP variables are saved with the snapshot
---------------------	--

Remarks

- This command acts globally, i.e. it can be applied in any script slot.
- In snapshot type 1 and 3, the value of 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](#)

9. 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 notes (C-2, C-1...C8)

See Also

[set_event_mark\(\)](#)

Events and MIDI: \$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

[change_velo\(\)](#)

[Events and MIDI](#): `$EVENT_NOTE`

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 the note event level and does not change any panorama settings in the instrument itself. It is also not related to any 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 millicents.

<ID-number>	The ID number of the note event to be changed.
<tune-amount>	The tune amount in millicents. 100000 equals 100 cents, 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 the note event level and does not change any tune settings in the instrument itself. It is also not related to any 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 every played 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

[change_note\(\)](#)

[Events and MIDI](#): `$EVENT_VELOCITY`

change_vol()

change_vol(<ID-number>,<volume>,<relative-bit>)

Change the volume of a specific note event in millidecibels

<ID-number>	The ID number of the note event to be changed
--------------------------	---

<volume>	The volume change in millidecibels
-----------------------	------------------------------------

<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 the 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).

Example

```
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>	The ID number of the event to be ungrouped
<bit-mark>	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\(\)](#)

[Events and MIDI](#): \$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

[get_event_ids\(\)](#)

Events and MIDI: **\$EVENT_STATUS_INACTIVE**, **\$EVENT_STATUS_NOTE_QUEUE**, **\$EVENT_STATUS_MIDI_QUEUE**

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 wheel 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_out\(\)](#)

get_event_ids()

get_event_ids(<array-name>)

Fills the specified array with all active event IDs

<array-name>	array to be filled with active event IDs
--------------	--

Remarks

- The command overwrites all existing values as long as there are active events, and writes 0 if no events are active anymore. If there are more active events than array indices, the array will be filled until it is full, ignoring the remaining event IDs.
- If there are less active events than array indices, the array will be filled from the beginning with all IDs followed by one index with its value set to 0.

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\(\)](#)

get_event_mark()

get_event_mark(<ID-number>, <bit-mark>)

Check if the specified event belongs to a specific event group (returns 1 if the bitmark is set, 0 otherwise)

<ID-number>	The ID number of the event to be checked
<bit-mark>	The bit-mark from \$MARK_1 to \$MARK_28

Examples

```
on note
  if ($EVENT_NOTE mod 12 = 0)
    set_event_mark($EVENT_ID, $MARK_1)
  end if
end on

on release
  if (get_event_mark($EVENT_ID, $MARK_1) = 1)
    message("You've played a C.")
  else
    message("")
  end if
end on
```

A rather long-winded way to check if you released a C.

See Also

[set_event_mark\(\)](#)

get_event_par()

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

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

<ID-number>	The ID number of the event
<parameter>	The event parameter, either one of four freely assignable event parameters: \$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_REL_VELOCITY \$EVENT_PAR_SOURCE \$EVENT_PAR_PLAY_POS \$EVENT_PAR_ZONE_ID (use with caution, refer to below)

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()`.

Examples

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

The same functionality as `message($EVENT_NOTE)`

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

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

```
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>, <index>)

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

<ID-number>	The ID number of the note event
<parameter>	Can be either \$EVENT_PAR_ALLOW_GROUP or \$EVENT_PAR_CUSTOM
<index>	When used with \$EVENT_PAR_ALLOW_GROUP: the group index. When used with \$EVENT_PAR_CUSTOM: the index (0 to 15).

Remarks

- `get_event_par_arr()` is a special form (or to be more precise, it's the array variant) of `get_event_par()`. It can be used to retrieve the group allow state of a specific event (returning 1 if the specified group is allowed and 0 if it's disallowed). It can also be used to retrieve a value when using more than the standard four event parameters when using \$EVENT_PAR_CUSTOM.

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\(\)](#)

Events and MIDI: \$EVENT_PAR_ALLOW_GROUP, %GROUPS_AFFECTED, \$EVENT_PAR_CUSTOM

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 information with `get_event_par()`, refer to 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\(\)](#)

redirect_output()

redirect_output(<ID-number>, <output-type>, <index>)

Routes the audio signal of the specified event to a specific output or bus.

<ID-number>	The ID number of the event to be routed.
<output-type>	Can be one of the following: \$OUTPUT_TYPE_DEFAULT The audio signal of the event is routed to the default instrument output. \$OUTPUT_TYPE_MASTER_OUT The audio signal of the event is routed directly to the output channel specified with <index>. The audio signal will not be affected by any instrument effect. \$OUTPUT_TYPE_AUX_OUT The audio signal of the event is routed directly to the Aux channel specified with <index>. The audio signal will not be affected by any instrument effect. \$OUTPUT_TYPE_BUS_OUT The audio signal of the event is routed to the instrument bus specified with <index>.
<index>	Specifies the output channel, aux channel or instrument bus, depending on <output-type>. Has no effect when <output-type> is set to \$OUTPUT - TYPE - DEFAULT.

Remarks

- When using **redirect_output()**, the output selection of a group is completely ignored.

Examples

```
on init
  declare $new_id_0
  declare $new_id_1
  declare $new_id_2
end on

on note
  ignore_event($EVENT_ID)

  $new_id_0 := play_note($EVENT_NOTE, $EVENT_VELOCITY, 0, -1)
  $new_id_1 := play_note($EVENT_NOTE+4, $EVENT_VELOCITY, 0, -1)
  $new_id_2 := play_note($EVENT_NOTE+7, $EVENT_VELOCITY, 0, -1)

  redirect_output($new_id_0, $OUTPUT_TYPE_BUS_OUT, 0)
  redirect_output($new_id_1, $OUTPUT_TYPE_BUS_OUT, 1)
  redirect_output($new_id_2, $OUTPUT_TYPE_BUS_OUT, 2)
end on
```

Creating a major triad and routing each chord tone to an instrument bus.

set_event_mark()

set_event_mark(<ID-number>, <bit-mark>)

Assign the specified event to a specific event group

<ID-number> The ID number of the event to be grouped

<bit-mark> 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, as 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\(\)](#)

Events and MIDI: `$EVENT_ID`, `$ALL_EVENTS`, `$MARK_1 ... $MARK_28`

set_event_par()

set_event_par(<ID-number>, <parameter>, <value>)

Assign a parameter to a specific event

<ID-number>	The ID number of the event
<parameter>	The event parameter, either one of four freely assignable event parameters: \$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_REL_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()**.
- If you need access to more than four custom parameters, use **set_event_par_arr()** with **\$EVENT_PAR_CUSTOM**

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)***

```
on init
  message("")
  declare ui_switch $switch

  declare ui_label $midiChan1 (1,1)
  declare ui_label $midiChan2 (1,1)
  declare ui_label $midiChan3 (1,1)
  declare ui_label $midiChan4 (1,1)
  declare ui_label $midiChan5 (1,1)
  declare ui_label $midiChan6 (1,1)
  declare ui_label $midiChan7 (1,1)
  declare ui_label $midiChan8 (1,1)
  declare ui_label $midiChan9 (1,1)
  declare ui_label $midiChan10 (1,1)
  declare ui_label $midiChan11 (1,1)
  declare ui_label $midiChan12 (1,1)
```

```

declare ui_label $midiChan13 (1,1)
declare ui_label $midiChan14 (1,1)
declare ui_label $midiChan15 (1,1)
declare ui_label $midiChan16 (1,1)

declare %midiChans[16]
  %midiChans[0] := get_ui_id($midiChan1)
  %midiChans[1] := get_ui_id($midiChan2)
  %midiChans[2] := get_ui_id($midiChan3)
  %midiChans[3] := get_ui_id($midiChan4)
  %midiChans[4] := get_ui_id($midiChan5)
  %midiChans[5] := get_ui_id($midiChan6)
  %midiChans[6] := get_ui_id($midiChan7)
  %midiChans[7] := get_ui_id($midiChan8)
  %midiChans[8] := get_ui_id($midiChan9)
  %midiChans[9] := get_ui_id($midiChan10)
  %midiChans[10] := get_ui_id($midiChan11)
  %midiChans[11] := get_ui_id($midiChan12)
  %midiChans[12] := get_ui_id($midiChan13)
  %midiChans[13] := get_ui_id($midiChan14)
  %midiChans[14] := get_ui_id($midiChan15)
  %midiChans[15] := get_ui_id($midiChan16)
end on

on release
  if ($switch=1)
    set_event_par($EVENT_ID, $EVENT_PAR_REL_VELOCITY, 127)
  end if

set_control_par_str(%midiChans[$MIDI_CHANNEL], $CONTROL_PAR_TEXT, get_event_par($EVENT_ID, $EVENT_PAR_REL_VELOCITY))
end on

```

Release velocity within an MPE context

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>,<index>)	
Assign an event parameter array to a specific event	
<ID-number>	The ID number of the note event
<parameter>	Can be one of the following: • \$EVENT_PAR_ALLOW_GROUP • \$EVENT_PAR_CUSTOM • \$EVENT_PAR_MOD_VALUE_ID
<value>	When used with \$EVENT_PAR_ALLOW_GROUP: the allow state for the group (1 or 0) When used with \$EVENT_PAR_CUSTOM: the value of the specified index When used with \$EVENT_PAR_MOD_VALUE_ID: the modulation value in the range from -1000000 to +1000000
<index>	When used with \$EVENT_PAR_ALLOW_GROUP: the group index When used with \$EVENT_PAR_CUSTOM: the index (0 to 15) When used with \$EVENT_PAR_MOD_VALUE_ID: the ID as set in the modulation assignment in KONTAKT with "from script" being selected as the modulation source (0 to 1000).

Remarks

- `set_event_par_arr()` is the array variant of `set_event_par()`. You can use it to set the group allow state of a specific event, or if you need to access more than four event parameters. You can also use it to set up event specific modulations in KONTAKT.

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 the first group is always played.

```
on init
  declare const $CUSTOM_EVENT_PAR_4 := 4
end on

on note
  set_event_par_arr...
  ($EVENT_ID,$EVENT_PAR_CUSTOM,$ENGINE_UPTIME,$CUSTOM_EVENT_PAR_4)
end on

on release
  message(get_event_par_arr...
  ($EVENT_ID,$EVENT_PAR_CUSTOM,$CUSTOM_EVENT_PAR_4))
end on
```

Simple implementation of \$EVENT_PAR_CUSTOM

```
on note
  if ($EVENT_NOTE = 60)
    set_event_par_arr($EVENT_ID,$EVENT_PAR_MOD_VALUE_ID,500000,1)
```

```
    end if  
end on
```

Only middle C (Note Nr. 60) will be processed any modulation assignment with "from script" selected as modulation source and "1" as the ID.

See Also

[allow_group\(\)](#)

[disallow_group\(\)](#)

[get_event_par_arr\(\)](#)

[set_event_par\(\)](#)

Events and MIDI: \$EVENT_PAR_ALLOW_GROUP

10. 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

Remarks

- This command does not work with arrays of real numbers.

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 as 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\(\)](#)

Events and MIDI: `%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.

Remarks

- This command does not work with arrays of real numbers.

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.

<array-variable>	The array to be sorted
-------------------------------	------------------------

<direction>	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\(\)](#)

11. 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. index of the first instrument group is 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

[disallow_group\(\)](#)

[set_event_par_arr\(\)](#)

Events and MIDI: \$ALL_GROUPS, \$EVENT_PAR_ALLOW_GROUP

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. index of the first instrument group is 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

[allow_group\(\)](#)

[set_event_par_arr\(\)](#)

[Events and MIDI](#): \$ALL_GROUPS, \$EVENT_PAR_ALLOW_GROUP

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 a value of 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.

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. index of the first instrument group is 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

[allow_group\(\)](#)

[disallow_group\(\)](#)

[find_group\(\)](#)

[output_channel_name\(\)](#)

Events and MIDI: \$ALL_GROUPS, \$NUM_GROUPS

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 a UI and `persistence_changed` callback.
- It is recommended to not use the `purge_group()` command in the callback of an automatable control.
- It is now possible to supply an async ID to the `purge_group()` function and get a return in the `async_complete` callback.

Examples

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

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

on async_complete
  if (get_purge_state(0) = 0)
    message("Group is purged")
  else
    message("Group is not purged")
  end if
end on
```

Unloading all samples of the first group

See Also

[get_purge_state\(\)](#)

12. Time-Related Commands

change_listener_par()

change_listener_par(<signal-type>, <parameter>)

Changes the parameters of the **on listener** callback. It 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\(\)](#)

[Callbacks and UI: \\$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\(\)](#)

[Transport and Timing](#): \$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 maximum 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. Triggering will occur even when transport is stopped.

See Also

[change_listener_par\(\)](#)

[Callbacks and UI: \\$NI_SIGNAL_TYPE](#)

stop_wait()

stop_wait(<callback-ID>, <parameter>)

Stops wait commands in the specified callback

<callback-ID> The callback's ID number in which the wait commands will be stopped

<parameter> **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 in succession resulting in multiple arpeggios.

See Also

[wait\(\)](#)

[wait_ticks\(\)](#)

[Callbacks and UI](#): Callback Type Variables and Constants

reset_ksp_timer

reset_ksp_timer

Resets the KSP timer (\$KSP_TIMER) to zero

Remarks

- Note that the `$KSP_TIMER` variable, due to its 32-bit signed nature, will reach its limit after 2147483648 microseconds, or roughly 35 minutes and 47 seconds.
- 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

[Transport and Timing](#): \$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, note that due to its 32-bit signed nature it will not return correct values if specified number of ticks at the current tempo exceeds 2147483648 microseconds, or roughly 35 minutes and 47 seconds.

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\(\)](#)

[Transport and Timing](#): \$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\(\)](#)

[Transport and Timing](#): `$DURATION_QUARTER`

wait_async()

wait_async(<asyncID>)

Waits until the async command identified by the <asyncID> is finished.

Remarks

When performing multiple operations it is also possible to collect them together and then calling the `wait_async()` function on the collection. When the operations are collected in this manner they will be calculated in one block resulting in a performance gain. If the async operation is not in the pipeline anymore or invalid, there is no wait and the script continues.

Examples

```
wait_async(set_engine_par($ENGINE_PAR_EFFECT_TYPE, ... $EFFECT_TYPE_CHORUS, -1, 2, 1))
```

Performing a single operation

```
%asyncid[0] := async_operation
%asyncid[1] := another_async_operation
...
%asyncid[x] := last_async_operation

$i := 0
while($i < num_elements(%asyncid))
    wait_async(%asyncid[$i])
    inc($i)
end while
```

Performing multiple operations

See also

General: `$NI_ASYNC_EXIT_STATUS`, `$NI_ASYNC_ID`

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\(\)](#)

13. User Interface Commands

add_menu_item()

add_menu_item(<variable>, <text>, <value>)

Create a menu entry

<variable>	The variable name of the menu UI control
<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

[get_menu_item_str\(\)](#)

[get_menu_item_value\(\)](#)

[get_menu_item_visibility\(\)](#)

[set_menu_item_str\(\)](#)

[set_menu_item_visibility\(\)](#)

[ui_menu](#)

Specific: `$CONTROL_PAR_SELECTED_ITEM_IDX`, `$CONTROL_PAR_NUM_ITEMS`

add_text_line()

add_text_line(<variable>, <text>)

Add a new text line in the specified label without erasing existing text

<variable>	The variable name of the label UI control
<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>, <generic>)

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

<ui-ID>	The ID number of the level meter UI control. You can retrieve the ID number with <code>get_ui_id()</code> .
<group>	The index of the group you want 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 from 0 to 15 to set the channel.
<generic>	Can be one of the following: • NI_LEVEL_METER_MAIN: used to access main effects • NI_LEVEL_METER_GROUP: used to access the group effects • NI_LEVEL_METER_INSERT: used to access the insert effects • 0-15: used to access individual buses

Remarks

- The level meters can be attached to the output of an instrument bus and the instrument main output. They can also be attached to compressor and limiter effects to display gain reduction data, with the ability to set min and max display values and inverting the display by using `$CONTROL_PAR_RANGE_MIN` and `$CONTROL_PAR_RANGE_MAX`.

Examples

```
on init
  declare $slot_idx := 0
  declare $bus_idx := 0
  declare $group_idx := 0
  declare $channel := 0

  declare ui_label $InstOutput (1,1)
  declare ui_label $BusOutput (1,1)
  declare ui_label $MainFX (1,1)
  declare ui_label $BusFX (1,1)
  declare ui_label $GroupFX (1,1)
  declare ui_label $empty (1,1)

  declare ui_level_meter $inst_output_lvl
  attach_level_meter(get_ui_id($inst_output_lvl), -1, -1, $channel, -1)

  declare ui_level_meter $bus_output_lvl
  attach_level_meter(get_ui_id($bus_output_lvl), -1, -1, $channel, $bus_idx)

  declare ui_level_meter $mainfx_output_lvl
  attach_level_meter(get_ui_id($mainfx_output_lvl), -1, $slot_idx, $channel, -2)

  declare ui_level_meter $busfx_output_lvl
  attach_level_meter(get_ui_id($busfx_output_lvl), -1, $slot_idx, $channel, $bus_idx)

  declare ui_level_meter $groupfx_output_lvl
```

```
attach_level_meter(get_ui_id($groupfx_output_lvl),$group_idx,$slot_idx,$channel,-1)
```

```
end on
```

Various level meters

See Also

[ui_level_meter](#)

Specific: \$CONTROL_PAR_BG_COLOR, \$CONTROL_PAR_OFF_COLOR,
\$CONTROL_PAR_ON_COLOR, \$CONTROL_PAR_OVERLOAD_COLOR,
\$CONTROL_PAR_PEAK_COLOR, \$CONTROL_PAR_VERTICAL, \$CONTROL_PAR_RANGE_MIN,
\$CONTROL_PAR_RANGE_MAX

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 name of the waveform display UI control.
<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 waveform display UI control 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.` operator 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 a zone named "Test" to the waveform display, also showing the zone's slices and a table.

See Also

[set_ui_wf_property\(\)](#)

[get_ui_wf_property\(\)](#)

[ui_waveform](#)

Zone and Slice Functions: [find_zone\(\)](#)

Specific: [Waveform Flag Constants](#), [Waveform Property Constants](#)

fs_get_filename()

fs_get_filename(<ui-ID>, <return-parameter>)

Return the filename of the last selected file in the file selector UI control.

<ui-ID>	The ID number of the file selector UI control. You can retrieve the the ID number with <code>get_ui_id()</code> .
----------------------	---

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

Remarks

- This command is only available in the UI callback of `ui_file_selector`

See Also

[fs_navigate\(\)](#)

[ui_file_selector](#)

fs_navigate()

fs_navigate(<ui-ID>, <direction>)

Jump to the next/previous file in the file selector UI control and trigger its callback.

<ui-ID>	The ID number of the file selector UI control. You can retrieve the ID number with <code>get_ui_id()</code> .
----------------------	---

<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
--------------------------	--

Remarks

- This command is only available in a UI callback
- It will always call the UI callback of the `ui_file_selector` it is pointed to

See Also

[fs_get_filename\(\)](#)

[ui_file_selector](#)

get_control_par()

get_control_par(<ui-ID>, <control-parameter>)

Retrieve various parameters of the specified UI control

<ui-ID>	The ID number of the UI control. You can retrieve the ID number with <code>get_ui_id()</code>
<control-parameter>	Parameter of the UI control we wish to retrieve, i.e. <code>\$CONTROL_PAR_WIDTH</code>

Remarks

- `get_control_par()` comes in three additional flavors:
- `get_control_par_arr()` for working with array-based controls (i.e. retrieving values from a particular `ui_table` index)
- `get_control_par_str()` for working with strings (i.e. retrieving text from `ui_label` or automation name from `ui_slider`)
- `get_control_par_str_arr()` (i.e. retrieving automation name of particular `ui_xy` cursor)

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\(\)](#)

General: `$CONTROL_PAR_KEY_SHIFT`, `$CONTROL_PAR_KEY_ALT`,
`$CONTROL_PAR_KEY_CONTROL`

get_font_id()

get_font_id(<file-name>)

Returns a font ID from an image file; the font ID can be used on any control that has dynamic text elements

<file-name>	Name of the image file, without extension. The image has to be in PNG format and reside in the "pictures" subfolder of the resource container.
-------------	--

Remarks

- This command is only available in the init callback.
- The images need to be formatted in a special way to be interpreted correctly as custom fonts. All characters need to be placed side-by-side, following the Windows-1252 character set, with a fully red (#FF0000) pixel at the top left of every character frame. Also, alpha layer needs to contain only one color.

Examples

```
on init
  declare ui_text_edit @textEdit
  set_control_par(get_ui_id(@textEdit), $CONTROL_PAR_FONT_TYPE, ...
    get_font_id("Font1"))
end on
```

Using a custom font on a ui_text_edit control

See Also

[set_control_par\(\)](#)

General: \$CONTROL_PAR_FONT_TYPE

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 UI control. You can retrieve the ID number with <code>get_ui_id()</code> .
------------------------	---

<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

[add_menu_item\(\)](#)

[get_menu_item_value\(\)](#)

[get_menu_item_visibility\(\)](#)

[get_menu_item_str\(\)](#)

[set_menu_item_value\(\)](#)

[set_menu_item_visibility\(\)](#)

Specific: `$CONTROL_PAR_SELECTED_ITEM_IDX`, `$CONTROL_PAR_NUM_ITEMS`

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 UI control. You can retrieve the ID number with <code>get_ui_id()</code> .
<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

[add_menu_item\(\)](#)

[get_menu_item_str\(\)](#)

[get_menu_item_visibility\(\)](#)

[set_menu_item_str\(\)](#)

[set_menu_item_value\(\)](#)

[set_menu_item_visibility\(\)](#)

Specific: `$CONTROL_PAR_SELECTED_ITEM_IDX`, `$CONTROL_PAR_NUM_ITEMS`

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 UI control. You can retrieve the ID number with <code>get_ui_id()</code> .
<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 $visibility
    declare ui_button $value
end on

on ui_control ($visibility)
    set_menu_item_visibility (get_ui_id($menu),$visibility))
end on

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

Clicking on Visibility button shows or hides the second menu entry, while clicking on Value button shows the visibility state of that same menu entry.

See Also

[add_menu_item\(\)](#)

[get_menu_item_str\(\)](#)

[get_menu_item_value\(\)](#)

[set_menu_item_str\(\)](#)

[set_menu_item_value\(\)](#)

[set_menu_item_visibility\(\)](#)

Specific: \$CONTROL_PAR_SELECTED_ITEM_IDX, \$CONTROL_PAR_NUM_ITEMS

get_ui_id()

get_ui_id(<variable>)

Retrieve the ID number of a 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 waveform UI control
<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\(\)](#)

[Zone and Slice Functions: find_zone\(\)](#)

Specific: [Waveform Flag Constants](#), [Waveform Property Constants](#)

hide_part()

hide_part(<variable>, <hide-mask>)

Hide specific parts of user interface controls

<variable>	The variable name of the UI control
<hide-mask>	Bitmask of visibility states for various parts of UI controls, consisting of the following constants: \$HIDE_PART_BG (background of knobs, labels, value edits and tables) \$HIDE_PART_VALUE (value of knobs and tables) \$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. This is also possible with other UI elements.

See Also

Specific: `$CONTROL_PAR_HIDE`, `$HIDE_PART_NOthing`, `$HIDE_WHOLE_CONTROL`

load_performance_view()

load_performance_view(<filename>)

Loads a performance view file (NCKP) that was created in the Creator Tools GUI Designer

<filename> The filename of the NCKP file, without extension, as a string (in quotation marks)

Remarks

- Only one performance view file can be loaded per script slot
- This command is only available in the init callback
- This command cannot be used alongside **make_perfview()**
- The performance view file should be in the performance_view subfolder of the resource container
- All contained controls are then accessible as if they were declared and set up in KSP; variable names can be identified in Creator Tools
- More information in the Creator Tools documentation

Examples

```
on init
    load_performance_view("performanceView")
end on

on ui_control ($testButton)
    if ($testButton = 0)
        set_control_par(get_ui_id($testSlider), $CONTROL_PAR_HIDE,
$HIDE_PART_WHOLE_CONTROL)
    else
        set_control_par(get_ui_id($testSlider), $CONTROL_PAR_HIDE,
$HIDE_PART_NOTHING)
    end if
end on
```

Loads a performance view file and then defines some basic behavior involving two of the contained controls

make_perfview

make_perfview

Activates the performance view for the respective script

Remarks

- `make_perfview` is only available in the init callback.
- Cannot be used alongside the `load_performance_view()` command.

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\(\)](#)

[set_ui_width_px\(\)](#)

[set_ui_color\(\)](#)

move_control()

move_control(<variable>, <x-position>, <y-position>)

Position UI elements in the standard KONTAKT grid

<variable>	The variable 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

See Also

[move_control_px\(\)](#)

General: `$CONTROL_PAR_HIDE`

move_control_px()

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

Position UI elements in pixels

<variable>	The variable 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.
- In order to match grid size to pixel position, the following formulas can be used:
- X position: $((\text{grid_value} - 1) * 92) + 66$
- Y position: $((\text{grid_value} - 1) * 21) + 2$
- Width (to be used with `$CONTROL_PAR_WIDTH`): $(\text{grid_value} * 92) - 5$
- Height (to be used with `$CONTROL_PAR_HEIGHT`): $(\text{grid_value} * 21) - 3$

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 position. This might be useful for reference.

See Also

[move_control\(\)](#)

General: `$CONTROL_PAR_POS_X`, `$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 variable name of the UI control
<text>	The info text to be displayed

Remarks

The text string used can contain a maximum of 320 characters.

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\(\)](#)

General:

\$CONTROL_PAR_HELP

set_control_par()

set_control_par(<ui-ID>, <control-parameter>, <value>)

Change various parameters of the specified UI control

<ui-ID>	The ID number of the UI control. You can retrieve the ID number with <code>get_ui_id()</code>
<control-parameter>	Parameter of the UI control we wish to retrieve, i.e. <code>\$CONTROL_PAR_WIDTH</code>
<value>	The (integer) value

Remarks

- `set_control_par_str()` is a variation of the command for usage with text strings.

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 also have to specify its position in pixels once you enter "pixel-mode".

```
on init
  declare ui_label $test (1,1)
  set_control_par_str(get_ui_id($test),$CONTROL_PAR_TEXT,"This is Text")
  set_control_par(get_ui_id($test),$CONTROL_PAR_TEXT_ALIGNMENT,1)
end on
```

Set and center text in labels

See Also

[get_control_par\(\)](#)

[get_ui_id\(\)](#)

set_control_par_arr()

set_control_par_arr(<ui-ID>, <control-parameter>, <value>, <index>)

Change various parameters of an element within an array-based UI control, e.g. cursors in the XY pad

<ui-ID>	The ID number of the UI control. You can retrieve the ID number with <code>get_ui_id()</code>
<control-parameter>	Parameter of the UI control we wish to retrieve, e.g. <code>\$CONTROL_PAR_AUTOMATION_ID</code>
<value>	The (integer) value
<index>	The element index

Remarks

- `set_control_par_str_arr()` is a variation of the command for usage with text strings.

Examples

```
on init
  make_perfview
  set_ui_height_px(350)

  declare ui_xy ?myXY[4]
  declare $xyID
  $xyID := get_ui_id(?myXY)

  set_control_par_arr($xyID, $CONTROL_PAR_AUTOMATION_ID, 0, 0)
  set_control_par_arr($xyID, $CONTROL_PAR_AUTOMATION_ID, 1, 1)
  set_control_par_arr($xyID, $CONTROL_PAR_AUTOMATION_ID, 2, 2)
  set_control_par_arr($xyID, $CONTROL_PAR_AUTOMATION_ID, 3, 3)

  set_control_par_str_arr($xyID, $CONTROL_PAR_AUTOMATION_NAME, ...
    "Cutoff", 0)
  set_control_par_str_arr($xyID, $CONTROL_PAR_AUTOMATION_NAME, ...
    "Resonance", 1)
  set_control_par_str_arr($xyID, $CONTROL_PAR_AUTOMATION_NAME, ...
    "Delay Pan", 2)
  set_control_par_str_arr($xyID, $CONTROL_PAR_AUTOMATION_NAME, ...
    "Delay Feedback", 3)
end on
```

Setting automation IDs and names of an XY pad with two cursors

See Also

General: `$CONTROL_PAR_AUTOMATION_ID`, `$CONTROL_PAR_AUTOMATION_NAME`

Specific: `$CONTROL_PAR_CURSOR_PICTURE`, `$HIDE_PART_CURSOR`

set_knob_defval()

set_knob_defval(<variable>, <value>)

Assign a default value to a knob to which the knob is reset when pressing [Ctrl] (Windows) or [Cmd] (macOS) + clicking the knob.

Remarks

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

Examples

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

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

Assigning default values to a knob and slider

See Also

General: `$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

General: [\\$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

General: \$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 UI control. You can retrieve the ID number with <code>get_ui_id()</code> .
<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

[add_menu_item\(\)](#)

[get_menu_item_str\(\)](#)

[get_menu_item_value\(\)](#)

[get_menu_item_visibility\(\)](#)

[set_menu_item_value\(\)](#)

[set_menu_item_visibility\(\)](#)

Specific: `$CONTROL_PAR_SELECTED_ITEM_IDX`, `$CONTROL_PAR_NUM_ITEMS`

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 UI control. You can retrieve the ID number with <code>get_ui_id()</code> .
<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

[add_menu_item\(\)](#)

[get_menu_item_str\(\)](#)

[get_menu_item_value\(\)](#)

[get_menu_item_visibility\(\)](#)

[set_menu_item_str\(\)](#)

[set_menu_item_visibility\(\)](#)

Specific: `$CONTROL_PAR_SELECTED_ITEM_IDX`, `$CONTROL_PAR_NUM_ITEMS`

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 UI control. You can retrieve the ID number with <code>get_ui_id()</code> .
<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, 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

[add_menu_item\(\)](#)

[get_menu_item_str\(\)](#)

[get_menu_item_value\(\)](#)

[get_menu_item_visibility\(\)](#)

[set_menu_item_str\(\)](#)

[set_menu_item_value\(\)](#)

Specific: `$CONTROL_PAR_SELECTED_ITEM_IDX`, `$CONTROL_PAR_NUM_ITEMS`

set_table_steps_shown()

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

Changes the number of displayed columns in a 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 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
```

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 sizes

```
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. This is also possible with other UI elements.

See Also

[add_text_line\(\)](#)

[set_control_par\(\)](#): `set_control_par_str()`

General: \$CONTROL_PAR_TEXT

set_ui_color()

set_ui_color(<hex value>)

Set the main background color of the performance view

<hex value>

The hexadecimal color value in the following format:

9ff0000h {red}

The **9** at the start lets KONTAKT know the value is a number.

The **h** at the end indicates that it is a hexadecimal value.

Remarks

- Can be used in all callbacks.

Examples

```
on init
  make_perfview
  set_ui_color(90000000h)
end on
```

Creates a black interface

See Also

[set_ui_height\(\)](#)

[set_ui_height_px\(\)](#)

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 750)
-----------------------	--

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
```

See Also

[set_ui_height\(\)](#)

[set_ui_height_px\(\)](#)

set_ui_width_px()

set_ui_width_px(<width>)

Set the width of a script performance view in pixels

<width> The width of the script in pixels (633 to 1000)

Remarks

- Only possible in the init callback.

Examples

```
on init
  make_perfview
  set_ui_height_px(750)
  set_ui_width_px(1000)
end on
```

Making a performance view with the largest possible dimensions

See Also

[set_ui_height_px\(\)](#)

set_ui_wf_property()

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

Sets different properties for the waveform control

<variable>	The variable of the waveform UI control
<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 a zone named "Test" to the waveform display and shows a play cursor within the waveform as long as you play a note

See Also

[get_ui_wf_property\(\)](#)

[ui_waveform](#)

[attach_zone\(\)](#)

[Zone and Slice Functions:](#) [find_zone\(\)](#)

[Specific:](#) [Waveform Flag Constants](#), [Waveform Property Constants](#)

14. 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. It can be either 1 (key pressed) or 0 (key released).

Remarks

- `get_key_triggerstate()` only works 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_key_type(<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

```
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 a built-in 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 will 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\(\)](#)

15. 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 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. It 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 buses, this parameter specifies the actual bus: \$NI_BUS_OFFSET + [0-15] one of the 16 buses 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 send effect slots

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. It 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 buses, this parameter specifies the actual bus: \$NI_BUS_OFFSET + [0-15] one of the 16 buses 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

get_voice_limit()

get_voice_limit(<voice-type>)

Returns the voice limit for the Time Machine Pro mode of the source module

<voice-type>

The voice type, can be one of the following:

\$NI_VL_TMPRO_STANDARD {Standard Mode}

\$NI_VL_TMRPO_HQ {High Quality Mode}

Examples

```
on init
  declare ui_label $label (3,2)

  add_text_line($label,"Standard Voice Limit: " & ...
  get_voice_limit($NI_VL_TMPRO_STANDARD))

  add_text_line($label,"HQ Voice Limit: " & ...
  get_voice_limit($NI_VL_TMRPO_HQ))
end on
```

Displaying TM Pro voice limits

See Also

[set_voice_limit\(\)](#)

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

General: \$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 engine parameter to be modified, e.g. \$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 . Buses and Main FX also reside on Instruments level, so you must set <group> to -1 if you want to address a bus.
<slot>	The slot index (zero-based) of the specified parameter. This 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: \$NI_SEND_BUS : Send Effect \$NI_INSERT_BUS : Insert Effect \$NI_MAIN_BUS : Main Effect For buses, this parameter specifies the actual bus: \$NI_BUS_OFFSET + [0-15] one of the 16 buses 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
```

A knob controls the 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 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

set_voice_limit()

set_voice_limit(<voice-type>, <value>)

Sets the voice limit for the Time Machine Pro mode of the source module

<voice-type>	The voice type, can be one of the following: \$NI_VL_TMPRO_STANDARD {Standard Mode} \$NI_VL_TMRPO_HQ {High Quality Mode}
---------------------------	--

<value>	The voice limit of the Time Machine Pro mode
----------------------	--

Remarks

- Changing voice limits is an asynchronous operation. This means, that one cannot reliably access the newly allocated voices immediately after instantiation. To resolve this, the `set_voice_limit()` command returns an **\$NI_ASYNC_ID** and triggers the `on async_complete` callback.

Examples

```
on init
  declare ui_value_edit $Voices (1,8,1)
  make_persistent($Voices)

  declare $change_voices_id

end on

on ui_control ($Voices)
  $change_voices_id := set_voice_limit($NI_VL_TMPRO_STANDARD,$Voices)
end on

on async_complete
  if ($NI_ASYNC_ID = $change_voices_id)
    message("New TPro Std Voice Limit: " &
get_voice_limit($NI_VL_TMPRO_STANDARD))
  end if
end on
```

Changing TM Pro voice limits

See Also

[get_voice_limit\(\)](#)

16. Zone Commands

General Information

User zones are a special kind of zone that allow for zone creation and manipulation “on the fly” and can be used to allow user interaction with the sampled content within an instrument (for example in conjunction with sample drag-and-drop). These zones must be declared via script in the **on init** callback.

When a user zone is created the mapping is set to 0 on all zone parameters by default (root key, high velocity, high note, low note etc...). Therefore, the zone will not show in the mapping editor’s normal view (it will be listed and present in the list view).

Note that some of the functions listed below only work on user zones, while some also work on every zone.

get_loop_par()

get_loop_par(<zone_id>, <loop-index>, <parameter>)

Returns the loop parameters of a zone

<zone_id>	The ID of the zone
<loop-index>	The index number of the loop
<parameter>	The following parameters are available: \$LOOP_PAR_MODE \$LOOP_PAR_START \$LOOP_PAR_LENGTH \$LOOP_PAR_XFADE \$LOOP_PAR_COUNT \$LOOP_PAR_TUNING

Remarks

- `get_loop_par()` works on every loop from every zone
- This function runs synchronously

Examples

```
message(get_loop_par($myZoneId, 0, $LOOP_PAR_MODE))
```

get_sample()

get_sample(<zone-id>, <return-parameter>)

Returns paths, file names and extensions of samples.

<zone-id>	The ID of the zone
<return-parameter>	The following parameters are available: \$NI_FILE_NAME \$NI_FILE_FULL_PATH \$NI_FILE_FULL_PATH_OS \$NI_FILE_EXTENSION

Remarks

- `get_sample()` works on every zone
- This function runs synchronously

Examples

```
message(get_sample(%NI_USER_ZONE_IDS[0], $NI_FILE_NAME))
```

See Also

Sample Parameters: \$NI_FILE_NAME, \$NI_FILE_FULL_PATH, \$NI_FILE_FULL_PATH_OS, \$NI_FILE_EXTENSION

get_zone_par()

get_zone_par(<zone-id>, <parameter>)

Returns the zone parameters

<zone-id>	The ID of the zone
<parameter>	The following parameters are available: \$ZONE_PAR_HIGH_KEY \$ZONE_PAR_LOW_KEY \$ZONE_PAR_HIGH_VELO \$ZONE_PAR_LOW_VELO \$ZONE_PAR_ROOT_KEY \$ZONE_PAR_FADE_LOW_KEY \$ZONE_PAR_FADE_HIGH_KEY \$ZONE_PAR_FADE_LOW_VELO \$ZONE_PAR_FADE_HIGH_VELO \$ZONE_PAR_VOLUME \$ZONE_PAR_PAN \$ZONE_PAR_TUNE \$ZONE_PAR_GROUP \$ZONE_PAR_SAMPLE_START \$ZONE_PAR_SAMPLE_END \$ZONE_PAR_SAMPLE_MOD_RANGE

Remarks

- `get_zone_par()` works on every zone
- This function runs synchronously

Examples

```
get_zone_par(%NI_USER_ZONE_IDS[0], $ZONE_PAR_PAN)
```

is_zone_empty()

is_zone_empty(<zone-ID>)

Returns 1 if a zone is empty (has no sample), otherwise returns 0

<zone-ID>

The ID of the zone

Examples

```
message("Zone empty status: " & is_zone_empty(%NI_USER_ZONE_IDS[0]))
```

set_loop_par()

set_loop_par(<zone-id>, <loop-index>, <parameter>, <value>)

Sets the loop parameters of a user zone

<zone-id>	The ID of the zone
<loop-index>	The index number of the loop
<parameter>	The following parameters are available: \$LOOP_PAR_MODE \$LOOP_PAR_START \$LOOP_PAR_LENGTH \$LOOP_PAR_XFADE \$LOOP_PAR_COUNT \$LOOP_PAR_TUNING
<value>	The value of the loop parameter

Remarks

- `set_loop_par()` only works in user zone loops
- When executed in the `init` callback, this function runs synchronously and returns -1
- When executed outside the `init` callback, this function runs asynchronously and returns an async ID

Examples

```
wait_async(set_loop_par(%NI_USER_ZONE_IDS[0], 0, $LOOP_PAR_MODE, $SampleLoopOnA))
```


set_num_user_zones()

set_num_user_zones(<number_of_user_zones>)

Creates empty user zones

<number_of_user_zones>

Defines the number of user zones to be created. **%NI_USER_ZONE_IDS** is the array of size **<number_of_user_zones>** with all the user zone IDs.

Remarks

- A maximum of 512 user zones per instrument can be created
- User zones are shown with a different color in the mapping editor
- User zones cannot be modified from the mapping editor
- In order to manipulate the user zones, the IDs stored in the **%NI_USER_ZONE_IDS** array should be used, instead of the hardcoded zone IDs

Examples

```
on init
...
    set_num_user_zones(2)
    set_zone_par(%NI_USER_ZONE_IDS[0], $ZONE_PAR_GROUP, 30)
    set_zone_par(%NI_USER_ZONE_IDS[1], $ZONE_PAR_GROUP, 31)
...
end on
```

set_sample

set_sample(<zone-id>, <sample-path>)

Sets the user sample in a zone

<zone-id>	The ID of the zone
<sample-path>	The sample path of the sample

Remarks

- `set_sample()` only works in user zones
- When executed in the `init` callback, this function runs synchronously and returns -1
- When executed outside the `init` callback, this function runs asynchronously and returns an async ID

Examples

```
on ui_control ($myMouseArea)
  if ($NI_MOUSE_EVENT_TYPE = $NI_MOUSE_EVENT_TYPE_DROP)
    if (num_elements(!NI_DND_ITEMS_AUDIO) = 1)
      $async_lock := 1
      wait_async(set_sample(%NI_USER_ZONE_IDS[0],
        !NI_DND_ITEMS_AUDIO[0]))
    end on
  end on
```

set_zone_par()

set_zone_par(<zone-id>, <parameter>, <value>)

Sets the user zone parameters

<zone-id>	The ID of the zone
<parameter>	The following flags are available: \$ZONE_PAR_HIGH_KEY \$ZONE_PAR_LOW_KEY \$ZONE_PAR_HIGH_VELO \$ZONE_PAR_LOW_VELO \$ZONE_PAR_ROOT_KEY \$ZONE_PAR_FADE_LOW_KEY \$ZONE_PAR_FADE_HIGH_KEY \$ZONE_PAR_FADE_LOW_VELO \$ZONE_PAR_FADE_HIGH_VELO \$ZONE_PAR_VOLUME \$ZONE_PAR_PAN \$ZONE_PAR_TUNE \$ZONE_PAR_GROUP \$ZONE_PAR_SAMPLE_START \$ZONE_PAR_SAMPLE_END \$ZONE_PAR_SAMPLE_MOD_RANGE
<value>	The value of the zone parameter

Remarks

- `set_zone_par()` only works in user zones
- When executed in the `init` callback, this function runs synchronously and returns -1
- When executed outside the `init` callback, this function runs asynchronously and returns an async ID

Examples

```
set_zone_par(%NI_USER_ZONE_IDS[0], $ZONE_PAR_GROUP, 0)
```

17. 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 file commands in KSP
- IR samples (.wav, .aif, .aiff, .ncw) 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 whether the command has completed its action, and whether or not the loading or saving was successful.

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/
```

Factory folder on Windows:

```
/C:/Program Files/Common Files/Native Instruments/Kontakt/
```

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

[on_async_complete](#)

General: `$NI_ASYNC_ID`, `$NI_ASYNC_EXIT_STATUS`

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 onwards. If the NKI belongs to an encoded library, it will point to its library folder. Otherwise, the user content directory is returned.

Example

```
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\(\)](#)

General: **\$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>	The array variable, this name must be present in the .nka file
<mode>	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 asynchronously. 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.
- .nka files loaded from the resource container should always have a newline character at the end of the file. If this last newline is missing, then KONTAKT will not know the file has ended and will continue to try and load other data from the resources container. Files generated by the **save_array()** command have this automatically, but if you are creating files manually, then this is something to take care of.

Example

```
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

[on async_complete](#)

[save_array\(\)](#)

General: [\\$NI_ASYNC_ID](#), [\\$NI_ASYNC_EXIT_STATUS](#)

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.

Example

```
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 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 located here: OS X: /Users/<username>/Documents/Native Instruments/Kontakt/ Windows: C:/Users/<username>/Documents/Native Instruments/Kontakt/
<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 buses, this parameter specifies the actual bus: \$NI_BUS_OFFSET + [0-15] one of the 16 buses

Remarks

- Please note that subfolders 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.

Example

```

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 if

```

end on

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

See Also

[get_folder\(\)](#)

[on async_complete](#)

General: [\\$NI_ASYNC_ID](#)

save_array()

save_array(<array-variable>, <mode>)

Saves an array to an external file, i.e. an .nka file

<array-variable>	The array to be saved
<mode>	0: A dialog window pops up, allowing you to save the .nka file. Can only be used in UI and PGS 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

[on_async_complete](#)

[load_array\(\)](#)

General: \$NI_ASYNC_ID, \$NI_ASYNC_EXIT_STATUS

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.

Example

```
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 file with a range specified by the `mf_set_export_area()` command.

<path>	The absolute path of the 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\(\)](#)

18. Music Information Retrieval

General Information

Music Information Retrieval (MIR) allows the extraction of meaningful features from audio files, such as pitch or the volume level of a sample. New KSP commands allow extraction of such parameters from samples via script. MIR functions are not asynchronous in the **init** callback (-1 as async ID), but asynchronous otherwise.

Note: the type detection functions listed below (Sample Type, Drum Type, and Instrument Type) are designed to process one-shot audio samples.

detect_pitch()

detect_pitch(<zone-id>, <pitch-result>)

Returns a real value representing the fundamental frequency of an audio sample, in semitones and cents. If detection fails, the function will set <pitch-result> to `~NI_DETECT_PITCH_INVALID`.

<zone-id>

The ID of the zone

<pitch-result>

The MIDI note value of the detected pitch

detect_loudness()

detect_loudness(<zone-id>, <loudness-result>)

Returns a real value representing the loudness of an audio sample in dB. Loudness is measured according to the standard established by the International Telecommunication Union: *Algorithms to measure audio program loudness and true-peak audio level* - ITU-R BS.1770-4 (2015). If detection fails, the function will set <loudness-result> to `~NI_DETECT_LOUDNESS_INVALID`.

<zone-id>

The ID of the zone

<loudness-result>

The detected loudness in dB

detect_peak()

detect_peak(<zone-id>, <peak-result>)	
Returns a real value representing peak level of an audio sample in dB. Peak is measured according to the standard established by the International Telecommunication Union: <i>Algorithms to measure audio program loudness and true-peak audio level - ITU-R BS.1770-4 (2015)</i> . If detection fails, the function will set <peak-result> to ~NI_DETECT_PEAK_INVALID .	
<zone-id>	The ID of the zone
<peak-result>	The detected peak level in dB

detect_rms()

detect_rms(<zone-id>, <rms-result>)

Returns a real value representing the RMS level of an audio sample in dB. If detection fails, the function will set <rms-result> to `~NI_DETECT_RMS_INVALID`.

<zone-id>	The ID of the zone
<rms-result>	The real value of the RMS level of the audio sample in dB

detect_sample_type()

detect_sample_type(<zone-id>, <sample-type-result>)

Assigns <sample-type-result> a \$NI_DETECT_SAMPLE_TYPE tag describing the whether an audio sample is a drum or an instrument. If detection fails, the function will set <sample-type-result> to \$NI_DETECT_SAMPLE_TYPE_INVALID.

<zone-id>

The ID of the zone

<sample-type-result>

The detected sample type, can be one of the following:

\$NI_DETECT_SAMPLE_TYPE_INVALID

\$NI_DETECT_SAMPLE_TYPE_INSTRUMENT

\$NI_DETECT_SAMPLE_TYPE_DRUM

detect_drum_type()

detect_drum_type(<zone-id>, <drum-type-result>)	
Assigns <drum-type-result> a \$NI_DETECT_DRUM_TYPE tag describing the drum type of an audio sample. Hint: use this function if detect_sample_type() determines that a given audio sample is of type \$NI_DETECT_SAMPLE_TYPE_DRUM. If detection fails, the function will set <drum-type-result> to ~NI_DETECT_DRUM_TYPE_INVALID.	
<zone-id>	The ID of the zone
<drum-type-result>	The detected drum type, can be one of the following: \$NI_DETECT_DRUM_TYPE_INVALID \$NI_DETECT_DRUM_TYPE_KICK \$NI_DETECT_DRUM_TYPE_SNARE \$NI_DETECT_DRUM_TYPE_CLOSED_HH \$NI_DETECT_DRUM_TYPE_OPEN_HH \$NI_DETECT_DRUM_TYPE_TOM \$NI_DETECT_DRUM_TYPE_CYMBAL \$NI_DETECT_DRUM_TYPE_CLAP \$NI_DETECT_DRUM_TYPE_SHAKER \$NI_DETECT_DRUM_TYPE_PERC_DRUM \$NI_DETECT_DRUM_TYPE_PERC_OTHER

detect_instrument_type()

detect_instrument_type(<zone-id>, <instrument-type-result>)	
Assigns <drum-type-result> a \$NI_DETECT_INSTRUMENT_TYPE tag describing the instrument type of an audio sample. Hint: use this function if detect_sample_type() determines that a given audio sample is of type \$NI_DETECT_SAMPLE_TYPE_INSTRUMENT. If detection fails, the function will set <instrument-type-result> to \$NI_DETECT_INSTRUMENT_TYPE_INVALID.	
<zone-id>	The ID of the zone
<instrument-type-result>	The detected instrument type, can be one of the following: \$NI_DETECT_INSTRUMENT_TYPE_INVALID \$NI_DETECT_INSTRUMENT_TYPE_BASS \$NI_DETECT_INSTRUMENT_TYPE_BOWED_STRING \$NI_DETECT_INSTRUMENT_TYPE_BRASS \$NI_DETECT_INSTRUMENT_TYPE_FLUTE \$NI_DETECT_INSTRUMENT_TYPE_GUITAR \$NI_DETECT_INSTRUMENT_TYPE_KEYBOARD \$NI_DETECT_INSTRUMENT_TYPE_MALLET \$NI_DETECT_INSTRUMENT_TYPE_ORGAN \$NI_DETECT_INSTRUMENT_TYPE_PLUCKED_STRING \$NI_DETECT_INSTRUMENT_TYPE_REED \$NI_DETECT_INSTRUMENT_TYPE_SYNTH \$NI_DETECT_INSTRUMENT_TYPE_VOCAL

Examples

```
wait_async(detect_pitch(%NI_USER_ZONE_IDS[0], ~pitch_result))
wait_async(set_zone_par(%NI_USER_ZONE_IDS[0], $ZONE_PAR_ROOT_KEY,
  real_to_int(round(~pitch_result))))
wait_async(set_zone_par(%NI_USER_ZONE_IDS[0], $ZONE_PAR_TUNE, real_to_int(100.0 *
  (round(~pitch_result) - ~pitch_result))))
```

Set the zone root key by rounding the pitch result to an integer value. Then set the zone tune to correct for the pitch offset.

```
wait_async(detect_sample_type(%NI_USER_ZONE_IDS[0], $sample_type))
if ($sample_type = $NI_DETECT_SAMPLE_TYPE_INSTRUMENT)
  wait_async(detect_instrument_type(%NI_USER_ZONE_IDS[0], $instrument_type))
else
  wait_async(detect_drum_type(%NI_USER_ZONE_IDS[0], $drum_type))
end if

if ($sample_type = $NI_DETECT_SAMPLE_TYPE_INSTRUMENT)
  if ($instrument_type = $NI_DETECT_INSTRUMENT_TYPE_BASS)
    set_text ($label_5, "Bass")
  end if
else
  if ($drum_type = $NI_DETECT_DRUM_TYPE_KICK)
    set_text ($label_5, "Kick")
  end if
end if
```

Detect whether a sample is of type instrument or drum, and detect the corresponding drum or instrument type.

19. 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, i.e. 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 asynchronous command, and thus the common asynchronous 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 activated 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.

Navigating 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 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 will need to put a MIDI file called "test.mid" into your KONTAKT Factory folder. Otherwise the defined error message will be displayed.

See Also

[on_async_complete](#)

[save_midi_file\(\)](#)

[mf_set_event_par\(\)](#)

[mf_get_event_par\(\)](#)

General: [\\$NI_ASYNC_ID](#), [\\$NI_ASYNC_EXIT_STATUS](#)

mf_set_export_area()

mf_set_export_area(<name>, <start-pos>, <end-pos>, <start-track>, <end-track>)

Defines the part of the 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 object.
<end-pos>	Defines the end position (in ticks) of the export area. Use -1 to set this to the end of the 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 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 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 will have to put a MIDI file called "test.mid" into your KONTAKT Factory folder.

See Also

[mf_insert_file\(\)](#)

[save_midi_file\(\)](#)

Specific: `$CONTROL_PAR_DND_BEHAVIOUR`

mf_set_num_export_areas()

```
mf_set_num_export_areas(<num>)
```

Sets the number of export areas, with a maximum of 512.

Remarks

- Area index 0 is set with the previously existing command `mf_set_export_area`.
- The contents of index 0 can be copied to other areas by calling `mf_copy_export_area`.

See Also

[mf_set_export_area\(\)](#)

[mf_copy_export_area\(\)](#)

mf_copy_export_area()

mf_copy_export_area(<index>)

Copies the content of MIDI export area 0 to the specified index.

Example

```
on init
  message("")
  make_perfview
  declare $i
  declare const $drag_areas := 4
  declare ui_label $label1 (1,1)
  declare ui_label $label2 (1,1)
  declare ui_label $label3 (1,1)
  declare ui_label $label4 (1,1)
  declare %labelID[$drag_areas]
  %labelID[0] := get_ui_id($label1)
  %labelID[1] := get_ui_id($label2)
  %labelID[2] := get_ui_id($label3)
  %labelID[3] := get_ui_id($label4)
  declare !midiTracks[$drag_areas]
  !midiTracks[0] := "Synth1"
  !midiTracks[1] := "Synth2"
  !midiTracks[2] := "Bass"
  !midiTracks[3] := "Melody"
  mf_insert_file(get_folder($GET_FOLDER_PATCH_DIR) & "/my_midi.mid",0,0,0)
  mf_set_num_export_areas($drag_areas+1)
  $i := 0
  while($i<$drag_areas)
    set_control_par(%labelID[$i], $CONTROL_PAR_DND_BEHAVIOUR, 1)
    set_control_par(%labelID[$i], $CONTROL_PAR_MIDI_EXPORT_AREA_IDX, $i+1)
    set_control_par_str(%labelID[$i], $CONTROL_PAR_TEXT, !
midiTracks[$i])
    mf_set_export_area(!midiTracks[$i], -1, -1, $i, $i)
    mf_copy_export_area($i+1)
    inc($i)
  end while
end on
```

Loads a MIDI file and distributes the content found in the first four MIDI channels to four separate MIDI areas.

See Also

[mf_set_export_area\(\)](#)

[mf_set_num_export_areas\(\)](#)

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 event will be inserted
<pos>	The position at which the event will be inserted, in ticks
<command>	Defines the command type of the 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 command
<byte2>	The second byte of the 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 parameters: \$EVENT_PAR_0 \$EVENT_PAR_1 \$EVENT_PAR_2 \$EVENT_PAR_3 Or the "built-in" parameters of a 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
<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\(\)](#)

Events and MIDI: **\$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 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\(\)](#)

[mf_get_id\(\)](#)

[save_midi_file\(\)](#)

Events and MIDI: **\$CURRENT_EVENT**

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 group 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\(\)](#)

Events and MIDI: [\\$ALL_EVENTS](#), [\\$CURRENT_EVENT](#)

[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\(\)](#)

Events and MIDI: \$ALL_EVENTS, \$CURRENT_EVENT

[by_marks\(\)](#)

[by_track\(\)](#)

[mf_set_mark\(\)](#)

[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\(\)](#)

Events and MIDI: [\\$ALL_EVENTS](#), [\\$CURRENT_EVENT](#)

[by_track\(\)](#)

[mf_set_mark\(\)](#)

[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 to the **by_marks()** command.

See Also

[mf_insert_file\(\)](#)

[mf_insert_event\(\)](#)

[mf_remove_event\(\)](#)

[Events and MIDI: \\$ALL_EVENTS, \\$CURRENT_EVENT](#)

[by_marks\(\)](#)

[mf_set_mark\(\)](#)

[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_prev\(\)](#)
[mf_get_prev_at\(\)](#)
[mf_get_last\(\)](#)
[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_prev()``mf_get_prev_at()``mf_get_last()``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>	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_num_tracks()`

`mf_get_prev()`

`mf_get_prev_at()`

`mf_get_last()`

`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_prev_at()`

`mf_get_last()`

`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_prev\(\)](#)

[mf_get_last\(\)](#)

[save_midi_file\(\)](#)

mf_get_num_tracks()

```
mf_get_num_tracks( )
```

Returns the number of tracks in the MIDI object

See Also

[mf_insert_file\(\)](#)

[mf_get_first\(\)](#)

[mf_get_next\(\)](#)

[mf_get_next_at\(\)](#)

[mf_get_prev\(\)](#)

[mf_get_prev_at\(\)](#)

[mf_get_last\(\)](#)

[save_midi_file\(\)](#)

20. Built-in Variables and Constants

General

\$CURRENT_SCRIPT_SLOT

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

%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**.

\$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, e.g. **\$NI_BUS_OFFSET + 2** will point to instrument bus 3.

\$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 for the current instrument.

\$PLAYED_VOICES_TOTAL

The amount of played voices for all instruments.

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.

Time Machine Pro Variables

User access for the two voice limits (Standard and High Quality) of the Time Machine Pro, to be used with `set_voice_limit()` and `get_voice_limit()`.

\$NI_VL_TMPRO_STANDARD**\$NI_VL_TMRPO_HQ****\$REF_GROUP_IDX**

Group index number of the currently viewed group.

Events and MIDI

\$ALL_GROUPS

Addresses all groups in the instrument when used 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

\$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_3`
- `$EVENT_PAR_VOLUME`
- `$EVENT_PAR_PAN`
- `$EVENT_PAR_TUNE`
- `$EVENT_PAR_NOTE`
- `$EVENT_PAR_VELOCITY`

Event Parameters to be used with `set_event_par_arr()` and `get_event_par_arr()`:

- `$EVENT_PAR_ALLOW_GROUP`
- `$EVENT_PAR_MOD_VALUE_ID`

Event Parameters 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 and 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!)

Event Parameters to be used with `get_event_par()`, multi script only:

- `$EVENT_PAR_MIDI_CHANNEL`
- `$EVENT_PAR_MIDI_COMMAND`
- `$EVENT_PAR_MIDI_BYTE_1`
- `$EVENT_PAR_MIDI_BYTE_2`

Event Parameters to be used with `mf_set_event_par()` and `mf_get_event_par()`:

- `$EVENT_PAR_POS`
- `$EVENT_PAR_NOTE_LENGTH`
- `$EVENT_PAR_ID`
- `$EVENT_PAR_TRACK_NR`

%EVENT_PAR

Array which contains values of `$EVENT_PAR_0` ... `$EVENT_PAR_3` along with `$EVENT_PAR_CUSTOM` indices 4-15, valid for `$EVENT_ID`.

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.

\$NOTE_HELD

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

%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

\$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)

\$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

%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, independent of the octave, is held. **0** otherwise. Due to this, the note number should be a value between 0 (C) and 11 (B).

Transport and Timing

\$DISTANCE_BAR_START

Returns the time of a note on message in microseconds from the beginning of the current bar with respect to the current tempo.

\$DURATION_BAR

Returns the duration in microseconds 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.

\$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 efficiency of script and not to make musical calculations, as musical calculations use the **\$ENGINE_UPTIME** timer.

\$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

\$SIGNATURE_NUM

Numerator of the current time signature, i.e. **4/4**

\$SIGNATURE_DENOM

Denominator of the current time signature, i.e. 4/**4**

Tempo Unit Variables

Used to control the unit parameter of time-related controls (e.g. Delay Time, Attack etc.) with engine parameter variables like \$ENGINE_PAR_DL_TIME_UNIT.

\$NI_SYNC_UNIT_ABS

\$NI_SYNC_UNIT_WHOLE

\$NI_SYNC_UNIT_WHOLE_TRIPLET

\$NI_SYNC_UNIT_HALF

\$NI_SYNC_UNIT_HALF_TRIPLET

\$NI_SYNC_UNIT_QUARTER

\$NI_SYNC_UNIT_QUARTER_TRIPLET

\$NI_SYNC_UNIT_8TH

\$NI_SYNC_UNIT_8TH_TRIPLET

\$NI_SYNC_UNIT_16TH

\$NI_SYNC_UNIT_16TH_TRIPLET

\$NI_SYNC_UNIT_32ND

\$NI_SYNC_UNIT_32ND_TRIPLET

\$NI_SYNC_UNIT_64TH

\$NI_SYNC_UNIT_64TH_TRIPLET

\$NI_SYNC_UNIT_256TH

\$NI_SYNC_UNIT_ZONE (Only applies to the Source Module Speed parameter)

%NOTE_DURATION[<note-number>]

Note length since note start in microseconds for each key.

\$NI_BAR_START_POSITION

Returns the start of current bar in ticks (at 960 PPQ) from the start of the host's song.

Callbacks and UI

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 that 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

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

\$NI_SIGNAL_TYPE

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

\$NI_KONTAKT_IS_HEADLESS

Returns **1** if the GUI of KONTAKT is not available and not loadable. Currently this is only possible with KONTAKT being used in MASCHINE+, i.e. with KONTAKT being used in a regular host, it will still return **0** even if the GUI is not visible.

Mathematical Constants

~NI_MATH_PI

Returns the mathematical constant pi (approx. 3.14159...)

~NI_MATH_E

Returns the mathematical constant e (approx. 2.71828...)

21. Control Parameters

General

\$CONTROL_PAR_NONE

Nothing will be applied to the control

\$CONTROL_PAR_HELP

Sets or returns the help text which is displayed in the info pane when hovering the control

\$CONTROL_PAR_PARENT_PANEL

Places a control to a panel. The value should be the UI ID of the panel

Size, Position, and Look

\$CONTROL_PAR_POS_X

Sets or returns the horizontal position in pixels

\$CONTROL_PAR_POS_Y

Sets or returns the vertical position in pixels

\$CONTROL_PAR_GRID_X

Sets or returns the horizontal position in grid units

\$CONTROL_PAR_GRID_Y

Sets or returns the vertical position in grid units

\$CONTROL_PAR_WIDTH

Sets or returns the width of the control in pixels

\$CONTROL_PAR_HEIGHT

Sets or returns the height of the control in pixels

\$CONTROL_PAR_GRID_WIDTH

Sets or returns the width of the control in grid units

\$CONTROL_PAR_GRID_HEIGHT

Sets or returns the height of the control in grid units

\$CONTROL_PAR_HIDE

Sets or returns 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 and tables)

\$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_PICTURE

Sets or returns 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/pictures), then in the KONTAKT pictures folder.

\$CONTROL_PAR_PICTURE_STATE

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

\$CONTROL_PAR_Z_LAYER

Sets or returns the Z layer position of the control. Controls can be placed in one of three layers. Within these layers they are then positioned by type, and then by declaration order.

0: Default layer. All controls are assigned to this layer by default

-1: Back layer. Controls in this layer are placed below the default layer

1: Front layer. Controls in this layer are placed on top of the default and back layers.

Z layer order by control type (from lowest level to highest):

File Selector

Waveform

Wavetable

Level Meter

Label

Knob

Slider

Switch

Button

Value Edit

Menu

Table

XY Pad

Text Edit

Mouse Area

Values

\$CONTROL_PAR_VALUE

Sets or returns the value

\$CONTROL_PAR_DEFAULT_VALUE

Sets or returns the default value

Text

\$CONTROL_PAR_TEXT

Sets or returns the control text, similar to `set_text()`

\$CONTROL_PAR_TEXTLINE

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

\$CONTROL_PAR_LABEL

Sets or returns the knob label, similar to `set_knob_label()`.

This is also the value/string published to the host when using automation.

This also works for switches.

\$CONTROL_PAR_UNIT

Sets or returns the knob unit, similar to `set_knob_unit()`

\$CONTROL_PAR_FONT_TYPE

Sets or returns the font type. Numbers 0 to 25 are used to select any of the 26 factory fonts, as shown below. Combine with `get_font_id()` to use custom fonts.

ID 0: The quick brown fox jumps over the lazy dog	ID 13: The quick brown fox jumps over the lazy dog
ID 1: The quick brown fox jumps over the lazy dog	ID 14: The quick brown fox jumps over the lazy dog
ID 2: The quick brown fox jumps over the lazy dog	ID 15: The quick brown fox jumps over the lazy dog
ID 3: The quick brown fox jumps over the lazy dog	ID 16: The quick brown fox jumps over the lazy dog
ID 4: The quick brown fox jumps over the lazy dog	ID 17: The quick brown fox jumps over the lazy dog
ID 5: The quick brown fox jumps over the lazy dog	ID 18: The quick brown fox jumps over the lazy dog
ID 6: The quick brown fox jumps over the lazy dog	ID 19: The quick brown fox jumps over the lazy dog
ID 7: The quick brown fox jumps over the lazy dog	ID 20: The quick brown fox jumps over the lazy dog
ID 8: The quick brown fox jumps over the lazy dog	ID 21: The quick brown fox jumps over the lazy dog
ID 9: The quick brown fox jumps over the lazy dog	ID 22: The quick brown fox jumps over the lazy dog
ID 10: The quick brown fox jumps over the lazy dog	ID 23: The quick brown fox jumps over the lazy dog
ID 11: The quick brown fox jumps over the lazy dog	ID 24: The quick brown fox jumps over the lazy dog
ID 12: The quick brown fox jumps over the lazy dog	ID 25: The quick brown fox jumps over the lazy dog

For responsive controls (buttons, switches and menus) the font can also be set separately for each of the control's states via the following control parameters:

- \$CONTROL_PAR_FONT_TYPE_ON
- \$CONTROL_PAR_FONT_TYPE_OFF_PRESSED
- \$CONTROL_PAR_FONT_TYPE_ON_PRESSED
- \$CONTROL_PAR_FONT_TYPE_OFF_HOVER
- \$CONTROL_PAR_FONT_TYPE_ON_HOVER

Not using any of the five additional state fonts will result in the default (`$CONTROL_PAR_FONT_TYPE`) being used for those states.

\$CONTROL_PAR_DISABLE_TEXT_SHIFTING

Deactivates text position shifting when clicking on buttons and switches

\$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

Automation

\$CONTROL_PAR_AUTOMATION_NAME

Sets or returns an automation name to a UI control when used with `set_control_par_str()`. **\$CONTROL_PAR_LABEL** can be used to set the automation value string. When assigning automation names to XY pad cursors, use the `set_control_par_str_arr()` command with this parameter.

\$CONTROL_PAR_ALLOW_AUTOMATION

Sets or returns if a UI control can be automated (**1**) or not (**0**). By default, automation is activated for all automatable controls. Can only be used in the init callback. Automation IDs can also be assigned to XY pad cursors using the `set_control_par_arr()` command.

\$CONTROL_PAR_AUTOMATION_ID

Sets or returns an automation ID to a UI control, in the range from 0 to 511. Can only be used in the init callback.

Automation IDs can only be assigned to automatable controls (sliders, switches, and knobs).

When assigning automation IDs to XY pad cursors, use the `set_control_par_arr()` command with this parameter.

Key Modifiers

\$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] (Windows) or [Opt] (macOS) key 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] (Windows) or [Cmd] (macOS) key was pressed (**0** otherwise) while clicking the UI control.

Menus and value edits are not supported.

Specific

Tables

\$NI_CONTROL_PAR_IDX

Returns the index of the table column that triggered the `on ui_control()` callback

Tables and Waveforms

\$CONTROL_PAR_BAR_COLOR

Sets or returns 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 or returns 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()`.

Mouse Area

\$CONTROL_PAR_DND_ACCEPT_AUDIO

Enables the mouse area to accept audio files.

\$CONTROL_PAR_DND_MIDI

Enables the mouse area to accept MIDI files.

\$CONTROL_PAR_DND_ACCEPT_ARRAY

Enables the mouse area to accept arrays.

All three flags can have one of the following values:

\$NI_DND_ACCEPT_NONE

\$NI_DND_ACCEPT_ONE

\$NI_DND_ACCEPT_MULTIPLE

\$CONTROL_PAR_RECEIVE_DRAG_EVENTS

Configures whether the mouse area's ui_control callback gets triggered just for the drop event (variable = 0) or also for drag events (variable = 1).
The ui_control callback has 2 built-in variables:

\$NI_MOUSE_EVENT_TYPE	Specifies the event type that triggered the callback and can have one of the following values: \$NI_MOUSE_EVENT_TYPE_DND_DROP \$NI_MOUSE_EVENT_TYPE_DND_DRAG
\$NI_MOUSE_OVER_CONTROL	Equals 1 if the mouse entered the mouse_area on a drag event Equals 0 if the mouse left the mouse_area on a drag event

Example

```
on ui_control ($aMouseArea)
  if ($NI_MOUSE_EVENT_TYPE = $NI_MOUSE_EVENT_TYPE_DROP)
    message(num_elements(!NI_DND_ITEMS_AUDIO))
  end if

  if ($NI_MOUSE_EVENT_TYPE = $NI_MOUSE_EVENT_TYPE_DRAG)
    message(num_elements(!NI_DND_ITEMS_AUDIO))
    message($MOUSE_OVER_CONTROL)
  end if
end on
```

Labels

\$CONTROL_PAR_DND_BEHAVIOUR

Sets or returns the drag and drop behavior for labels. 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. For more information on MIDI handling in KSP, refer to [MIDI Object Commands](#).

Value Edit

\$CONTROL_PAR_SHOW_ARROWS

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

Level Meters

\$CONTROL_PAR_BG_COLOR

Sets or returns 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)

\$CONTROL_PAR_RANGE_MIN \$CONTROL_PAR_RANGE_MAX

Sets the minimum and maximum display range of the level meters, with default ranges from 0 to 1000000.

If the minimum values is smaller than the maximum value, the display is inverted.

File Selector

\$CONTROL_PAR_BASEPATH

Sets or returns the basepath of the UI file selector. This control parameter can be used in any callback.

Be careful with the number of subfolders of the basepath as it might take too long to scan the sub file system.

\$CONTROL_PAR_COLUMN_WIDTH

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

\$CONTROL_PAR_FILEPATH

Sets or returns the actual path (full path of the file) of the UI file selector. The file path must be a subpath of the instrument's 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 or returns 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 will not 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.

Waveform

Waveform Flag Constants

To be used with **attach_zone()**. You can combine flag constants using the bit-wise **.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()`

<code>\$UI_WF_PROP_PLAY_CURSOR</code>	Sets or returns the play head position
<code>\$UI_WF_PROP_FLAGS</code>	Used to set new flag constants after the <code>attach_zone()</code> command is used
<code>\$UI_WF_PROP_TABLE_VAL</code>	Sets or returns the value of the indexed slice's table
<code>\$UI_WF_PROP_TABLE_IDX_HIGHLIGHT</code>	Highlights the indexed slice within the UI waveform
<code>\$UI_WF_PROP_MIDI_DRAG_START_NOTE</code>	Defines the start note for the MIDI drag and drop function

\$CONTROL_PAR_WF_VIS_MODE

Changes the way the waveform is drawn. Valid values:

`$NI_WF_VIS_MODE_1` (default)

`$NI_WF_VIS_MODE_2` (X-ray)

`$NI_WF_VIS_MODE_3` (X-ray filled)

\$CONTROL_PAR_BG_COLOR

Sets the background color of the waveform display

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_WAVE_COLOR

Sets or returns the color of the waveform

\$CONTROL_PAR_WAVE_ALPHA

Sets or returns the alpha channel (opacity) of the waveform.

Range: **0** (fully transparent) to **255** (fully opaque)

\$CONTROL_PAR_WAVE_CURSOR_COLOR

Sets or returns the color of the playback cursor

\$CONTROL_PAR_SLICEMARKERS_COLOR

Sets or returns the color of the slice markers

\$CONTROL_PAR_BG_ALPHA

Sets or returns the alpha channel (opacity) of the background of the widget.
Range: **0** (fully transparent) to **255** (fully opaque)

Wavetable**\$CONTROL_PAR_WT_ZONE**

Attaches a zone to a wavetable taking the zone idx as argument.

\$CONTROL_PAR_WT_VIS_MODE

Sets the mode of the wavetable widget. Can be set to the following values:

\$NI_WT_VIS_2D (2D, oscilloscope-style visualization, only showing the current wavetable position)

\$NI_WT_VIS_3D (3D visualization displaying the whole wavetable as well as the the current position)

\$CONTROL_PAR_PARALLAX_X

Sets the x-axis parallax of the wavetable control (only applicable to 3D mode)
Range: **-1000000** to **1000000**

\$CONTROL_PAR_PARALLAX_Y

Sets the y-axis parallax of the wavetable control (only applicable to 3D mode)
Range: **-1000000** to **1000000**

\$CONTROL_PAR_WAVETABLE_COLOR

Sets or returns the color of the whole wavetable

\$CONTROL_PAR_WAVETABLE_ALPHA

Sets or returns the alpha channel (opacity) of the whole wavetable
Range: **0** (fully transparent) to **255** (fully opaque)

\$CONTROL_PAR_BG_COLOR

Sets or returns the background color of the wavetable widget.

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_BG_ALPHA

Sets or returns the alpha channel (opacity) of the background of the widget.
Range: **0** (fully transparent) to **255** (fully opaque)

Additional Color and Alpha Parameters

To be paired with the ones above to create gradient effects; if not explicitly set, they inherit the value of their match from above, resulting in no gradient.

\$CONTROL_PAR_WAVE_END_COLOR	Sets or returns the color for the end of the gradient applied to the waveform
\$CONTROL_PAR_WAVE_END_ALPHA	Sets or returns the alpha channel (opacity) for the end of the gradient applied to the waveform
\$CONTROL_PAR_WAVETABLE_END_COLOR	Sets or returns the color for the end of the gradient applied to the wavetable
\$CONTROL_PAR_WAVETABLE_END_ALPHA	Sets or returns the alpha channel (opacity) for the end of the gradient applied to the wavetable

Slider**\$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 behavior.

XY Pad**\$CONTROL_PAR_MOUSE_BEHAVIOUR_X**

Mouse behavior, i.e the drag scale, of the x axis of all cursors

\$CONTROL_PAR_MOUSE_BEHAVIOUR_Y

Mouse behavior, i.e the drag scale, of the y axis of all cursors

\$CONTROL_PAR_MOUSE_MODE

Sets the way the XY pad responds to mouse clicks and drags.

0: Clicks anywhere other than on a cursor are ignored. Clicking on a cursor and dragging, sets new values respecting the usual **\$CONTROL_PAR_MOUSE_BEHAVIOR** settings.

1: Clicks anywhere on the XY pad are registered but don't change the values. Clicking anywhere and dragging, sets new values; the cursor moves parallel to the mouse cursor with distances scaled based on the **\$CONTROL_PAR_MOUSE_BEHAVIOR** settings.

2: Clicks anywhere on the XY pad are registered and immediately change the values, with the cursor immediately matching the mouse cursor. Clicking anywhere and dragging has a similar effect; the **\$CONTROL_PAR_MOUSE_BEHAVIOR** settings are ignored; cursor always follows mouse cursor one-to-one.

\$CONTROL_PAR_ACTIVE_INDEX

Sets and gets the index of the active cursor. Only relevant in multi-cursor set-ups. The **\$CONTROL_PAR_MOUSE_MODE** setting will influence how this parameter behaves:

Mouse Mode = **0** and **1**: the active cursor can only be changed manually, by setting this control parameter. Inactive cursors don't receive any clicks.

Mouse Mode = **2**: it is set automatically based on the last clicked cursor. Setting it manually from within the **ui_control callback** of the XY pad can result in unexpected results, but using it in other callbacks is fully encouraged and makes sense in many scenarios. The value is -1 when not clicking on any cursor.

The index can only be an even number (with the exception of the -1 value) that matches the index of the X axis of the cursor in the main array representing the XY control, e.g. the first cursor has an index of 0, the second one has an index of 2, etc

\$CONTROL_PAR_CURSOR_PICTURE

Sets the cursor image. Each cursor can have its own image set using the **set_control_par_str_arr()** command.

Using **\$CONTROL_PAR_PICTURE** with the XY pad will set the background image of the control.

The cursor images can have up to 6 frames, corresponding to the following states. Frame selection is automatic as with buttons/switches.

- 1: Inactive
- 2: Active
- 3: Inactive pressed
- 4: Active pressed
- 5: Inactive mouse over
- 6: Active mouse over

\$HIDE_PART_CURSOR

When used with **set_control_par_arr()**, this can be used to hide specific cursors in the XY pad. Below is a simple syntax example:

```
if($hide = 1)
  set_control_par_arr($id, $CONTROL_PAR_HIDE, $HIDE_PART_CURSOR, $index)
else
  set_control_par_arr($id, $CONTROL_PAR_HIDE, $HIDE_PART_NOTHING, $index)
end if
```

The index should be an even number that matches the index of the X axis of the cursor in the main array representing the XY control, so the first cursor has an index of 0, the second has an index of 2, and so on.

\$NI_CONTROL_PAR_IDX

Returns the index of the cursor that triggered the on **ui_control()** callback for the XY pad. Note that indices are always even numbers starting from 0, so the first cursor has an index of 0, the second has an index of 2, and so on.

\$NI_MOUSE_EVENT_TYPE

Returns the type of mouse event that triggered the on `ui_control()` callback for the XY pad. Can only be used within a on `ui_control()` callback.

The following file types are available:

`$NI_MOUSE_EVENT_TYPE_LEFT_BUTTON_DOWN` (click)

`$NI_MOUSE_EVENT_TYPE_LEFT_BUTTON_UP` (release)

`$NI_MOUSE_EVENT_TYPE_DRAG` (drag)

22. Engine Parameters

Instrument, Source and Amp Module

\$ENGINE_PAR_VOLUME

Instrument/group/bus volume

\$ENGINE_PAR_PAN

Instrument/group/bus panorama

\$ENGINE_PAR_PHASE_INVERT

Group phase invert

\$ENGINE_PAR_LR_SWAP

Group LR swap

\$ENGINE_PAR_TUNE

Instrument/group 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_SPEED_UNIT
 \$ENGINE_PAR_WT_POSITION
 \$ENGINE_PAR_WT_FORM
 \$ENGINE_PAR_WT_PHASE
 \$ENGINE_PAR_WT_PHASE_RAND
 \$ENGINE_PAR_WT_QUALITY
 \$NI_WT_QUALITY_LOFI
 \$NI_WT_QUALITY_MEDIUM
 \$NI_WT_QUALITY_HIGH
 \$NI_WT_QUALITY_BEST
 \$ENGINE_PAR_WT_FORM_MODE
 \$NI_WT_FORM_LINEAR
 \$NI_WT_FORM_SYNC1
 \$NI_WT_FORM_SYNC2
 \$NI_WT_FORM_SYNC3
 \$NI_WT_FORM_BENDP
 \$NI_WT_FORM_BENDM
 \$NI_WT_FORM_BENDMP
 \$NI_WT_FORM_PWM
 \$NI_WT_FORM_ASYMP
 \$NI_WT_FORM_ASYMM
 \$NI_WT_FORM_ASYMMP
 \$NI_WT_FORM_FLIP
 \$NI_WT_FORM_MIRROR
 \$NI_WT_FORM_QUANTIZE
 \$ENGINE_PAR_WT_INHARMONIC_MODE
 \$ENGINE_PAR_WT_INHARMONIC

\$ENGINE_PAR_POST_FX_SLOT

Sets the number of Post Amp Group effects.

\$ENGINE_PAR_OUTPUT_CHANNEL

Designates the output for the group or bus.

0 - 63 routes to one of KONTAKT's available 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 buses. Buses cannot be routed to other buses.

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

\$ENGINE_PAR_COMP_LINK

\$ENGINE_PAR_COMP_TYPE

\$NI_COMP_TYPE_CLASSIC

\$NI_COMP_TYPE_ENHANCED

\$NI_COMP_TYPE_PRO

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

\$ENGINE_PAR_FCOMP_HQ_MODE

\$ENGINE_PAR_FCOMP_LINK

Limiter

\$ENGINE_PAR_LIM_IN_GAIN

\$ENGINE_PAR_LIM_RELEASE

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

\$ENGINE_PAR_SCOMP_LINK

Supercharger GT

\$EFFECT_TYPE_SUPERGT
\$ENGINE_PAR_SUPERGT_TRIM
\$ENGINE_PAR_SUPERGT_HPF_MODE
\$NI_SUPERGT_HPF_MODE_OFF
\$NI_SUPERGT_HPF_MODE_100
\$NI_SUPERGT_HPF_MODE_300
\$ENGINE_PAR_SUPERGT_SATURATION
\$ENGINE_PAR_SUPERGT_SAT_MODE
\$NI_SUPERGT_SAT_MODE_MILD
\$NI_SUPERGT_SAT_MODE_MODERATE
\$NI_SUPERGT_SAT_MODE_HOT
\$ENGINE_PAR_SUPERGT_COMPRESS
\$ENGINE_PAR_SUPERGT_ATTACK
\$ENGINE_PAR_SUPERGT_RELEASE
\$ENGINE_PAR_SUPERGT_CHARACTER
\$ENGINE_PAR_SUPERGT_CHAR_MODE
\$NI_SUPERGT_CHAR_MODE_FAT
\$NI_SUPERGT_CHAR_MODE_WARM
\$NI_SUPERGT_CHAR_MODE_BRIGHT
\$ENGINE_PAR_SUPERGT_MIX
\$ENGINE_PAR_SUPERGT_CHANNEL_LINK_MODE
\$NI_SUPERGT_CHANNEL_LINK_MODE_STEREO
\$NI_SUPERGT_CHANNEL_LINK_MODE_DUAL_MONO
\$NI_SUPERGT_CHANNEL_LINK_MODE_MS

Transient Master

\$ENGINE_PAR_TR_INPUT
\$ENGINE_PAR_TR_ATTACK
\$ENGINE_PAR_TR_SUSTAIN
\$ENGINE_PAR_TR_SMOOTH

Transparent Limiter

\$EFFECT_TYPE_TRANS LIM
\$ENGINE_PAR_TRANS LIM_THRESHOLD
\$ENGINE_PAR_TRANS LIM_RELEASE
\$ENGINE_PAR_TRANS LIM_CEILING

ACBox

\$ENGINE_PAR_AC_NORMALVOLUME
\$ENGINE_PAR_AC_BRILLIANTVOLUME
\$ENGINE_PAR_AC_BASS
\$ENGINE_PAR_AC_TREBLE
\$ENGINE_PAR_AC_TONECUT
\$ENGINE_PAR_AC_TREMOLOSPEED
\$ENGINE_PAR_AC_TREMOLODEPTH
\$ENGINE_PAR_AC_MONO

Bass Invader

\$ENGINE_PAR_BASSINVADER_VOLUME
\$ENGINE_PAR_BASSINVADER_TREBLE
\$ENGINE_PAR_BASSINVADER_HI_MID
\$ENGINE_PAR_BASSINVADER_LO_MID
\$ENGINE_PAR_BASSINVADER_BASS
\$ENGINE_PAR_BASSINVADER_BOOST
\$ENGINE_PAR_BASSINVADER_MASTER
\$ENGINE_PAR_BASSINVADER_LO_CUT
\$ENGINE_PAR_BASSINVADER_MID_CONTOUR
\$ENGINE_PAR_BASSINVADER_HI_BOOST

Bass Pro

\$ENGINE_PAR_BASSPRO_GAIN
\$ENGINE_PAR_BASSPRO_BASS
\$ENGINE_PAR_BASSPRO_MID
\$ENGINE_PAR_BASSPRO_MIDFREQ
\$ENGINE_PAR_BASSPRO_TREBLE
\$ENGINE_PAR_BASSPRO_DRIVE
\$ENGINE_PAR_BASSPRO_MASTER
\$ENGINE_PAR_BASSPRO_GEQ_40
\$ENGINE_PAR_BASSPRO_GEQ_90
\$ENGINE_PAR_BASSPRO_GEQ_180
\$ENGINE_PAR_BASSPRO_GEQ_300
\$ENGINE_PAR_BASSPRO_GEQ_500
\$ENGINE_PAR_BASSPRO_GEQ_1K
\$ENGINE_PAR_BASSPRO_GEQ_2K
\$ENGINE_PAR_BASSPRO_GEQ_4K
\$ENGINE_PAR_BASSPRO_GEQ_10K
\$ENGINE_PAR_BASSPRO_GEQ_VOLUME
\$ENGINE_PAR_BASSPRO_ULTRALO
\$ENGINE_PAR_BASSPRO_ULTRAHI
\$ENGINE_PAR_BASSPRO_BRIGHT
\$ENGINE_PAR_BASSPRO_GEQ
\$ENGINE_PAR_BASSPRO_MONO

Cabinet

\$ENGINE_PAR_CB_SIZE
\$ENGINE_PAR_CB_AIR
\$ENGINE_PAR_CB_TREBLE
\$ENGINE_PAR_CB_BASS
\$ENGINE_PAR_CABINET_TYPE

HotSolo

\$ENGINE_PAR_HS_PRENORMAL
\$ENGINE_PAR_HS_PREOVERDRIVE
\$ENGINE_PAR_HS_BASS
\$ENGINE_PAR_HS_MID
\$ENGINE_PAR_HS_TREBLE
\$ENGINE_PAR_HS_MASTER
\$ENGINE_PAR_HS_PRESENCE
\$ENGINE_PAR_HS_DEPTH
\$ENGINE_PAR_HS_OVERDRIVE
\$ENGINE_PAR_HS_MONO

Jump

\$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

Twang

\$ENGINE_PAR_TW_VOLUME
\$ENGINE_PAR_TW_TREBLE
\$ENGINE_PAR_TW_MID
\$ENGINE_PAR_TW_BASS
\$ENGINE_PAR_TW_BRIGHT
\$ENGINE_PAR_TW_MONO

Van51

\$ENGINE_PAR_V5_PREGAINRHYTHM
\$ENGINE_PAR_V5_PREGAINLEAD
\$ENGINE_PAR_V5_BASS
\$ENGINE_PAR_V5_MID
\$ENGINE_PAR_V5_TREBLE
\$ENGINE_PAR_V5_POSTGAIN
\$ENGINE_PAR_V5_RESONANCE
\$ENGINE_PAR_V5_PRESENCE
\$ENGINE_PAR_V5_LEADCHANNEL
\$ENGINE_PAR_V5_HIGAIN
\$ENGINE_PAR_V5_BRIGHT
\$ENGINE_PAR_V5_CRUNCH
\$ENGINE_PAR_V5_MONO

Cat

\$ENGINE_PAR_CT_VOLUME
\$ENGINE_PAR_CT_DISTORTION
\$ENGINE_PAR_CT_FILTER
\$ENGINE_PAR_CT_BASS
\$ENGINE_PAR_CT_BALLS
\$ENGINE_PAR_CT_TREBLE
\$ENGINE_PAR_CT_TONE
\$ENGINE_PAR_CT_MONO

Cry Wah

\$ENGINE_PAR_CW_MONO
\$ENGINE_PAR_CW_PEDAL

Distortion

\$ENGINE_PAR_DRIVE
\$ENGINE_PAR_DAMPING
\$ENGINE_PAR_DISTORTION_TYPE
\$NI_DISTORTION_TYPE_TUBE
\$NI_DISTORTION_TYPE_TRANS

DStortion

\$ENGINE_PAR_DS_VOLUME
\$ENGINE_PAR_DS_TONE
\$ENGINE_PAR_DS_DRIVE
\$ENGINE_PAR_DS_BASS
\$ENGINE_PAR_DS_MID
\$ENGINE_PAR_DS_TREBLE
\$ENGINE_PAR_DS_MONO

Lo-Fi

\$ENGINE_PAR_BITS
\$ENGINE_PAR_FREQUENCY
\$ENGINE_PAR_NOISELEVEL
\$ENGINE_PAR_NOISECOLOR

Saturation

\$ENGINE_PAR_SHAPE
\$ENGINE_PAR_SHAPE_TYPE
\$NI_SHAPE_TYPE_CLASSIC
\$NI_SHAPE_TYPE_ENHANCED
\$NI_SHAPE_TYPE_DRUMS

Skreamer

\$ENGINE_PAR_SK_TONE
\$ENGINE_PAR_SK_DRIVE
\$ENGINE_PAR_SK_BASS
\$ENGINE_PAR_SK_BRIGHT
\$ENGINE_PAR_SK_MIX

Tape Saturator

\$ENGINE_PAR_TP_GAIN
\$ENGINE_PAR_TP_WARMTH
\$ENGINE_PAR_TP_HF_ROLLOFF
\$ENGINE_PAR_TP_QUALITY

Choral

\$ENGINE_PAR_CHORAL_RATE
\$ENGINE_PAR_CHORAL_MODE
\$NI_CHORAL_MODE_SYNTH
\$NI_CHORAL_MODE_ENSEMBLE
\$NI_CHORAL_MODE_DIMENSION
\$NI_CHORAL_MODE_UNIVERSAL
\$ENGINE_PAR_CHORAL_AMOUNT
\$ENGINE_PAR_CHORAL_VOICES
\$ENGINE_PAR_CHORAL_DELAY
\$ENGINE_PAR_CHORAL_WIDTH
\$ENGINE_PAR_CHORAL_FEEDBACK
\$ENGINE_PAR_CHORAL_SCATTER
\$ENGINE_PAR_CHORAL_INVERT_PHASE
\$ENGINE_PAR_CHORAL_MIX

Flair

\$ENGINE_PAR_FLAIR_MODE
 \$NI_FLAIR_MODE_STANDARD
 \$NI_FLAIR_MODE_THRU_ZERO
 \$NI_FLAIR_MODE_SCAN
 \$ENGINE_PAR_FLAIR_CHORD
 \$ENGINE_PAR_FLAIR_INVERT_PHASE
 \$ENGINE_PAR_FLAIR_RATE
 \$ENGINE_PAR_FLAIR_RATE_UNIT
 \$ENGINE_PAR_FLAIR_FEEDBACK
 \$ENGINE_PAR_FLAIR_AMOUNT
 \$ENGINE_PAR_FLAIR_WIDTH
 \$ENGINE_PAR_FLAIR_PITCH
 \$ENGINE_PAR_FLAIR_DAMPING
 \$ENGINE_PAR_FLAIR_VOICES
 \$ENGINE_PAR_FLAIR_DETUNE
 \$ENGINE_PAR_FLAIR_MIX
 \$ENGINE_PAR_FLAIR_OFFSET
 \$ENGINE_PAR_FLAIR_SCANMODE
 \$NI_FLAIR_SCANMODE_TRIANGLE
 \$NI_FLAIR_SCANMODE_SAW_UP
 \$NI_FLAIR_SCANMODE_SAW_DOWN

Phasis

\$ENGINE_PAR_PHASIS_RATE
 \$ENGINE_PAR_PHASIS_RATE_UNIT
 \$ENGINE_PAR_PHASIS_ULTRA
 \$ENGINE_PAR_PHASIS_AMOUNT
 \$ENGINE_PAR_PHASIS_CENTER
 \$ENGINE_PAR_PHASIS_STEREO
 \$ENGINE_PAR_PHASIS_SPREAD
 \$ENGINE_PAR_PHASIS_FEEDBACK
 \$ENGINE_PAR_PHASIS_MOD_MIX
 \$ENGINE_PAR_PHASIS_NOTCHES
 \$ENGINE_PAR_PHASIS_INVERT_PHASE
 \$ENGINE_PAR_PHASIS_INVERT_MOD_MIX
 \$ENGINE_PAR_PHASIS_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

AET Filter

\$ENGINE_PAR_EXP_FILTER_MORPH
\$ENGINE_PAR_EXP_FILTER_AMOUNT

Inverter

\$ENGINE_PAR_PHASE_INVERT
\$ENGINE_PAR_LR_SWAP

Send Levels

\$ENGINE_PAR_SENLEVEL_0
\$ENGINE_PAR_SENLEVEL_1
\$ENGINE_PAR_SENLEVEL_2
<...>
\$ENGINE_PAR_SENLEVEL_7

Stereo Modeller

\$ENGINE_PAR_STEREO
\$ENGINE_PAR_STEREO_PAN

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

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 Ladder and Daft filter types

\$ENGINE_PAR_FILTER_LADDER_HQ

High Quality mode for the Ladder 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

Replika Delay

\$ENGINE_PAR_RDL_TYPE

\$NI_REPLIKA_TYPE_MODERN

\$NI_REPLIKA_TYPE_TAPE

\$NI_REPLIKA_TYPE_VINTAGE

\$NI_REPLIKA_TYPE_DIFFUSION

\$NI_REPLIKA_TYPE_ANALOGUE

\$ENGINE_PAR_RDL_TIME

\$ENGINE_PAR_RDL_TIME_UNIT

\$ENGINE_PAR_RDL_FEEDBACK

\$ENGINE_PAR_RDL_LOWCUT

\$ENGINE_PAR_RDL_HIGHCUT

\$ENGINE_PAR_RDL_SATURATION

\$ENGINE_PAR_RDL_TAPEAGE

\$ENGINE_PAR_RDL_FLUTTER

\$ENGINE_PAR_RDL_QUALITY

\$ENGINE_PAR_RDL_DEPTH

\$ENGINE_PAR_RDL_RATE

\$ENGINE_PAR_RDL_TYPE

\$ENGINE_PAR_RDL_STEREO

\$ENGINE_PAR_RDL_NOISE

\$ENGINE_PAR_RDL_PINGPONG

\$ENGINE_PAR_RDL_AMOUNT

\$ENGINE_PAR_RDL_SIZE

\$ENGINE_PAR_RDL_DENSE

\$ENGINE_PAR_RDL_MODULATION

\$ENGINE_PAR_RDL_BBDTYPE

Legacy Delay

\$ENGINE_PAR_DL_TIME
 \$ENGINE_PAR_DL_TIME_UNIT
 \$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
 \$ENGINE_PAR_IRC_REVERSE
 \$ENGINE_PAR_IRC_ER_LR_BOUNDARY

Plate Reverb

\$ENGINE_PAR_PR_DECAY
 \$ENGINE_PAR_PR_LOWSHELF
 \$ENGINE_PAR_PR_HIDAMP
 \$ENGINE_PAR_PR_PREDELAY
 \$ENGINE_PAR_PR_STEREO

Reverb

\$ENGINE_PAR_RV2_TYPE
 \$NI_REVERB2_TYPE_ROOM
 \$NI_REVERB2_TYPE_HALL
 \$ENGINE_PAR_RV2_TIME
 \$ENGINE_PAR_RV2_SIZE
 \$ENGINE_PAR_RV2_DAMPING
 \$ENGINE_PAR_RV2_MOD
 \$ENGINE_PAR_RV2_DIFF
 \$ENGINE_PAR_RV2_PREDELAY
 \$ENGINE_PAR_RV2_HIGHCUT
 \$ENGINE_PAR_RV2_LOWSHELF
 \$ENGINE_PAR_RV2_STEREO
 \$ENGINE_PAR_RV2_TYPE

Legacy Reverb

\$ENGINE_PAR_RV_PREDELAY
\$ENGINE_PAR_RV_SIZE
\$ENGINE_PAR_RV_COLOUR
\$ENGINE_PAR_RV_STEREO
\$ENGINE_PAR_RV_DAMPING

Legacy Chorus

\$ENGINE_PAR_CH_DEPTH
\$ENGINE_PAR_CH_SPEED
\$ENGINE_PAR_CH_SPEED_UNIT
\$ENGINE_PAR_CH_PHASE

Legacy Flanger

\$ENGINE_PAR_FL_DEPTH
\$ENGINE_PAR_FL_SPEED
\$ENGINE_PAR_FL_SPEED_UNIT
\$ENGINE_PAR_FL_PHASE
\$ENGINE_PAR_FL_FEEDBACK
\$ENGINE_PAR_FL_COLOR

Legacy Phaser

\$ENGINE_PAR_PH_DEPTH
\$ENGINE_PAR_PH_SPEED
\$ENGINE_PAR_PH_SPEED_UNIT
\$ENGINE_PAR_PH_PHASE
\$ENGINE_PAR_PH_FEEDBACK

Gainer

\$ENGINE_PAR_GN_GAIN

Modulation

\$ENGINE_PAR_MOD_TARGET_INTENSITY

The intensity slider of a modulation assignment. This controls the modulation amount.

\$MOD_TARGET_INVERT_SOURCE

The Invert button of a modulation assignment. This inverts the modulation amount.

\$ENGINE_PAR_INTMOD_BYPASS

The bypass button of an internal modulator, e.g. AHDSR envelope, LFO

\$ENGINE_PAR_INTMOD_RETRIGGER

The Retrigger button of a modulation assignment. This restarts the envelope every time a note is received.

AHDSR

\$ENGINE_PAR_ATK_CURVE
 \$ENGINE_PAR_ATTACK
 \$ENGINE_PAR_ATTACK_UNIT
 \$ENGINE_PAR_HOLD
 \$ENGINE_PAR_HOLD_UNIT
 \$ENGINE_PAR_DECAY
 \$ENGINE_PAR_DECAY_UNIT
 \$ENGINE_PAR_SUSTAIN
 \$ENGINE_PAR_RELEASE
 \$ENGINE_PAR_RELEASE_UNIT
 \$ENGINE_PAR_ENV_AHD

DBD

\$ENGINE_PAR_DECAY1
 \$ENGINE_PAR_DECAY1_UNIT
 \$ENGINE_PAR_BREAK
 \$ENGINE_PAR_DECAY2
 \$ENGINE_PAR_DECAY2_UNIT
 \$ENGINE_PAR_ENV_DBD_EASY

Envelope Follower

\$ENGINE_PAR_ENVF_ATTACK
 \$ENGINE_PAR_ENVF_RELEASE
 \$ENGINE_PAR_ENVF_GAIN_BOOST
 \$ENGINE_PAR_ENVF_ADAPTION

LFO

For all LFOs:

\$ENGINE_PAR_INTMOD_FREQUENCY

\$ENGINE_PAR_INTMOD_FREQUENCY_UNIT

\$ENGINE_PAR_LFO_DELAY

\$ENGINE_PAR_LFO_DELAY_UNIT

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_RANDOM

\$ENGINE_PAR_LFO_NORMALIZE

Glide

\$ENGINE_PAR_GLIDE_COEF

\$ENGINE_PAR_GLIDE_COEF_UNIT

Module Types and Subtypes

\$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_NONE (Empty slot)
 \$EFFECT_TYPE_FILTER
 \$EFFECT_TYPE_SOLID_GEQ
 \$EFFECT_TYPE_COMPRESSOR
 \$EFFECT_TYPE_FB_COMP
 \$EFFECT_TYPE_LIMITER
 \$EFFECT_TYPE_BUS_COMP
 \$EFFECT_TYPE_SUPERGT
 \$EFFECT_TYPE_TRANS_MASTER
 \$EFFECT_TYPE_TRANSLIM
 \$EFFECT_TYPE_ACBOX
 \$EFFECT_TYPE_BASSINVADER
 \$EFFECT_TYPE_BASSPRO
 \$EFFECT_TYPE_CABINET
 \$EFFECT_TYPE_HOTSOLO
 \$EFFECT_TYPE_JUMP
 \$EFFECT_TYPE_TWANG
 \$EFFECT_TYPE_VAN51
 \$EFFECT_TYPE_CAT
 \$EFFECT_TYPE_CRYWAH
 \$EFFECT_TYPE_DISTORTION
 \$EFFECT_TYPE_DSTORTION
 \$EFFECT_TYPE_LOFI
 \$EFFECT_TYPE_SHAPER (Saturation)
 \$EFFECT_TYPE_SKREAMER
 \$EFFECT_TYPE_TAPE_SAT
 \$EFFECT_TYPE_REPLIKA
 \$EFFECT_TYPE_DELAY
 \$EFFECT_TYPE_IRC (Convolution)
 \$EFFECT_TYPE_PLATEREVERB
 \$EFFECT_TYPE_REVERB2
 \$EFFECT_TYPE_REVERB
 \$EFFECT_TYPE_CHORUS
 \$EFFECT_TYPE_FLANGER
 \$EFFECT_TYPE_PHASER
 \$EFFECT_TYPE_CHORAL
 \$EFFECT_TYPE_FLAIR
 \$EFFECT_TYPE_PHASIS
 \$EFFECT_TYPE_ROTATOR
 \$EFFECT_TYPE_AET_FILTER

\$ENGINE_PAR_EFFECT_TYPE

\$EFFECT_TYPE_GAINER

\$EFFECT_TYPE_INVERTER

\$EFFECT_TYPE_SEND_LEVELS

\$EFFECT_TYPE_SURROUND_PANNER

\$EFFECT_TYPE_STEREO (Stereo Modeller)

\$ENGINE_PAR_SEND_EFFECT_TYPE

Used to query the type of a send effect. Can be any of the following:

\$EFFECT_TYPE_NONE (Empty slot)

\$EFFECT_TYPE_REPLIKA

\$EFFECT_TYPE_DELAY

\$EFFECT_TYPE_IRC (Convolution)

\$EFFECT_TYPE_PLATEREVERB

\$EFFECT_TYPE_REVERB2

\$EFFECT_TYPE_REVERB

\$EFFECT_TYPE_CHORUS

\$EFFECT_TYPE_FLANGER

\$EFFECT_TYPE_PHASER

\$EFFECT_TYPE_GAINER

\$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

\$ENGINE_PAR_EFFECT_SUBTYPE

\$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_BBPB
 \$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_SINE
 \$LFO_TYPE_RECTANGLE
 \$LFO_TYPE_TRIANGLE
 \$LFO_TYPE_SAWTOOTH
 \$LFO_TYPE_RANDOM
 \$LFO_TYPE_MULTI

Group Start Options Query

Group Start Options Constants

\$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

23. Zone Parameters

Zone Parameters

These set the parameters for the user zones via KSP in the same manner and ranges as available on the mapping editor. They can be set with the `set_zone_par(<zone-id>, <parameter>, <value>)` function and retrieved with the `get_zone_par(<zone-id>, <parameter>)` function. When the zones are declared, all these parameters are set to 0 by default.

\$ZONE_PAR_HIGH_KEY

Sets the high key for the zone. Range: 0 to 127

\$ZONE_PAR_LOW_KEY

Sets the low key of the zone. Range: 0 to 127

\$ZONE_PAR_HIGH_VELO

Sets the maximum velocity response of the zone. Range: 1 to 127

\$ZONE_PAR_LOW_VELO

Sets the minimum velocity response of the zone. Range: 1 to 127

\$ZONE_PAR_ROOT_KEY

Sets the root key of the zone. Range: 0 to 127

\$ZONE_PAR_FADE_LOW_KEY

Optionally use this parameter to create zone crossfades. The value is set in the form of a distance to the `$ZONE_PAR_LOW_KEY`.

Range: $\$ZONE_PAR_HIGH_KEY - \$ZONE_PAR_LOW_KEY + 1$

\$ZONE_PAR_FADE_HIGH_KEY

Optionally use this parameter to create zone crossfades. The value is set in the form of a distance to the `$ZONE_PAR_HIGH_KEY`.

Range: $\$ZONE_PAR_HIGH_KEY - \$ZONE_PAR_LOW_KEY + 1$

\$ZONE_PAR_FADE_LOW_VELO

Optionally use this parameter to create zone crossfades. The value is set in the form of a distance to the `$ZONE_PAR_LOW_VELO`.

Range: $\$ZONE_PAR_HIGH_VELO - \$ZONE_PAR_LOW_VELO + 1$

\$ZONE_PAR_FADE_HIGH_VELO

Optionally use this parameter to create zone crossfades. The value is set in the form of a distance to the **\$ZONE_PAR_HIGH_VELO**.

Range: **\$ZONE_PAR_HIGH_VELO** - **\$ZONE_PAR_LOW_VELO** + 1

\$ZONE_PAR_VOLUME

Sets the volume of the zone. Range: -3600 to 3600

\$ZONE_PAR_PAN

Sets the panning of the zone. Range: -1000 to 1000

\$ZONE_PAR_TUNE

Sets the tuning of the zone. Range: -3600 to 3600

\$ZONE_PAR_GROUP

Sets the group of the user zone. By default a user zone is placed in group 0.

Examples

```
set_num_user_zones(4)
set_zone_par(%NI_USER_ZONE_IDS[0], $ZONE_PAR_GROUP, 30)
set_zone_par(%NI_USER_ZONE_IDS[1], $ZONE_PAR_GROUP, 31)
set_zone_par(%NI_USER_ZONE_IDS[2], $ZONE_PAR_GROUP, 72)
set_zone_par(%NI_USER_ZONE_IDS[3], $ZONE_PAR_GROUP, 73)
```

\$ZONE_PAR_SAMPLE_START

Sets the sample start value of the sample attached to the zone.

\$ZONE_PAR_SAMPLE_END

Sets the sample end value of the sample attached to the zone.

\$ZONE_PAR_SAMPLE_MOD_RANGE

User zone loop parameters work in the same manner as manually setting loops via the wave editor.

Loop Parameters

\$LOOP_PAR_MODE

The Loop Mode of the selected loop within the zone. Range: 0 to 4

0: Loop off

1: Loop until end, alternate off

2: Loop until end, alternate on

3: Loop until release, alternate off

4: Loop until release, alternate on

Examples

```
on ui_control($SampleLoopOn)
    wait_async(set_loop_par(%NI_USER_ZONE_IDS[2], 0, ...
        $LOOP_PAR_MODE, $SampleLoopOn))
end on
```

\$LOOP_PAR_START

The starting point in samples of the selected loop within the zone. If this parameter is not the loop will start at the beginning of the sample.

\$LOOP_PAR_LENGTH

The loop length in samples of the selected loop within the zone. If this parameter is not set the loop length will correspond to the entire sample.

\$LOOP_PAR_XFADE

The crossfade value in microseconds for the selected loop within the zone.

\$LOOP_PAR_COUNT

The number of times the selected loop within the zone will repeat. If this parameter is not set (or is set to 0), the loop will continue indefinitely.

\$LOOP_PAR_TUNING

Sets the tuning offset inside the loop area for the selected loop within the zone, applied on the first repeat of the loop, and for all successive repeats (as defined by **\$LOOP_PAR_COUNT**).

Sample Parameters

\$NI_FILE_NAME

The file name of a zone's sample (corresponds to the zone name)

\$NI_FILE_FULL_PATH

The full path of a zone's sample (same result as without the parameter)

\$NI_FILE_FULL_PATH_OS

The full OS path of a zone's sample (uses backslashes on Windows)

\$NI_FILE_EXTENSION

The file extension of a zone's sample (without the dot)

24. 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

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. This means, **SET_CONDITION** and **RESET_CONDITION** are not actually 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 three built-in symbols:

- **NO_SYS_SCRIPT_PEDAL**
- **NO_SYS_SCRIPT_RLS_TRIG**
- **NO_SYS_SCRIPT_GROUP_START**

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 that can be read/written by any script.

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 similar to a variable name; it can only contain letters and numbers and must not start with a number. It is 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 that 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 is 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 available 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 in 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 declared 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 Resources and Data folders next to the NKR file you just created.

The Resources folder is the place where you can store the files that an NKI can use, which 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 the 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, .aiff or .ncw 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.
- Check the performance_view folder for any .nckp files containing performance views created by Creator Tools.

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.

Changing FX from KSP

Introduction

Prior to KONTAKT 5.5, there was already the infrastructure in place to get info about the content of effect slots via engine parameter variables like `$ENGINE_PAR_EFFECT_TYPE` and built-in constants like `$EFFECT_TYPE_FILTER` (refer to Module Status Retrieval).

Starting with KONTAKT 5.5, it is also possible to change FX with the same set of built-in variables.

Example

```
on init
    set_engine_par($ENGINE_PAR_EFFECT_TYPE,$EFFECT_TYPE_FILTER,0,0,-1)
    set_engine_par($ENGINE_PAR_EFFECT_SUBTYPE,$FILTER_TYPE_LDR_LP4,0,0,-1)
end on
```

Inserts a 4 pole lowpass ladder filter into the first group slot

on async_complete callback

Changing FX slot contents is an asynchronous operation. This means, one cannot reliably access the newly instantiated effect immediately after instantiation. To resolve this, the command returns an `$NI_ASYNC_ID` and triggers the `on async_complete` callback. In addition, `wait_async()` command can be used to streamline this process.

Default Filter Type

Filters are somewhat special as they are effect types that feature subtypes. Since one can now instantiate a new filter from KSP without explicitly selecting its subtype, there is a need for a predefined default filter subtype. This is SV LP4.

Implications on Modulation and Automation assignments

When changing the contents of FX slots through KSP, it is expected that the handling of assigned automation and modulation is identical to performing the same action using KONTAKT's GUI.

- When changing a slot's effect type or removing it entirely, all modulation and automation assignments are also removed. Specifically to modulators, if the removed assignments are the only ones of a certain one (i.e., if the modulator is not assigned to other targets as well), the modulator itself is also removed.
- When changing a slot's effect subtype (only applies to filters), everything is left unchanged. It is accepted that in certain cases, one may end up with "orphaned" modulation assignments as it is the case right now; e.g., when having modulation assigned to a parameter that is no longer available, like Resonance or Gain.

Changing Modulator Subtypes

Using the same commands described above, one can also change the subtype of internal modulators. Specifically, one could switch between envelope types (AHDSR, Flex and DBD), or LFO types (Rectangle, Triangle, Sawtooth, Random and Multi). A modulator cannot be inserted or removed. Its Type (LFO, Envelope, Step Modulator, Envelope Follower and Glide) cannot be changed either.

Special Cases

There are two effect types that cannot be set from KSP:

- Surround Panner
- AET filter

The Advanced Engine Tab

The Advanced Engine tab can be a useful tool for debugging and measuring the performance of your scripts.

While the Engine tab (a sub-tab of the Expert tab in the Browser Pane) can provide a useful display of performance statistics, the advanced version gives higher accuracy to things like CPU usage, and also displays information on multiple instances of KONTAKT when it is used as a plug-in.

Displaying the Advanced Engine Tab

As mentioned earlier, the Engine tab is a sub section of the Expert tab, which can be found in the Browser Pane.

- To access the Advanced Engine tab, hold the [Alt] (Windows) or [Opt] (macOS) key while clicking on the Engine tab.
- To return to the main Engine tab, just click on the Engine tab again with no keys held.

Instance Overview

If you are running multiple instances of KONTAKT as a plug-in in a DAW or host, each instance will be given an entry in this section. If you are using KONTAKT in standalone, only the current instance will be displayed.

There are five performance statistics you can view here:

- **CPU:** displays the current CPU load in percent (at a higher resolution than the other CPU readouts in KONTAKT) as well as the highest recorded peak CPU level (displayed in parenthesis). You can reset the high peak by re-initializing the KONTAKT instance by clicking on the Engine Restart (!) button.
- **Voices:** displays the total number of voices currently in use by the KONTAKT instance.
- **Voices killed:** displays the total number of voices that have been killed due to CPU overload (displayed on the left) and DFD overload (displayed on the right).
- **Process Buffer:** displays the current audio buffer size in samples.
- **Events:** displays the total number of events currently in the event queue. While a voice is the equivalent to a sample being played back, an event is more closely related to MIDI note messages being processed by the engine. For example, a single event could produce 3 voices, if there are 3 samples mapped to a single note. Additionally, if you are holding a MIDI key event though the triggered sample has finished playback, the voice will terminate, but the event will remain in the queue. As such, this display can be useful for tracking down events that are hanging, as these are not always audible in the way that hanging voices would be.

Total

The lower section displays the total performance statistics for all KONTAKT instances currently loaded. It has the following parameters:

- **Voices** and **Voices killed:** like the displays in the Instance Overview, but a total for all instances.
- **DFD load:** if you are playing Instruments that use DFD mode, this measures their hard disk access. It is essentially a more accurate version of the Disk meter in KONTAKT's Main Header.
- **DFD memory:** a measurement of how much RAM is being used to process the DFD stream.
- **DFD requests:** the total number of requests made by KONTAKT to read data from the hard disk.

25. Multi Script

General Information

The multi script utilizes 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 as instrument scripts are saved with the instrument, multi scripts are saved with the multi. In relation to GUIs, everything is identical with the instrument script. The scripts are stored in a folder called "multiscripts", which resides next to the already existing "scripts" folder inside the "presets" folder:

/Native Instruments/Kontakt/presets/multiscripts

The multi script has four callback types: **on midi_in**, **on persistence_changed**, **on init** 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 as opposed to the instrument script.

On the instrument level, you can retrieve the event IDs of notes only, i.e., **\$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, these commands will not work in the multi script.

And most important of all, remember that the multi script is nothing more than a MIDI processor, whereas 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 events

Remarks

- Like `ignore_event()`, `ignore_midi` is a very "strong" command. Keep in mind that `ignore_midi` will ignore all incoming events.
- If you just want to change the MIDI channel and/or any of the bytes, you can also use `set_event_par()`.

Example

```
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

Example

```
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()`.

Example

```
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 – note that you also have to supply the correct note off commands

See Also

[set_event_par\(\)](#)

Events and MIDI: \$EVENT_PAR_MIDI_CHANNEL, \$EVENT_PAR_MIDI_COMMAND,
\$EVENT_PAR_MIDI_BYTE_1, \$EVENT_PAR_MIDI_BYTE_2

Multi Script Command Arguments

\$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 message, always in the range from 0 to 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

26. New Features

KONTAKT 6.7.0

New Features

- New snapshot types for saving only the persistent KSP variables with snapshots,
- Engine parameters for the following new effects:
 - Bass Pro
 - Bass Invader

KONTAKT 6.6.0

New Features

- New functionality enabling "from script" modulation assignments in KONTAKT with new built-in variable to be used with `set_event_par_arr()`: `$EVENT_PAR_MOD_VALUE_ID`
- New command for redirecting the audio signal of an event to an output or bus: `redirect_output()`
- New built-in variables to be used with `redirect_output()`:
`$OUTPUT_TYPE_DEFAULT`, `$OUTPUT_TYPE_MASTER_OUT`, `$OUTPUT_TYPE_AUX_OUT`,
`$OUTPUT_TYPE_BUS_OUT`
- The `ui_level_meter` can be attached to gain reduction data (i.e. from compressor and limiter effects).
- New built-in variables for setting the zoom level with gain reduction meters:
`$CONTROL_PAR_RANGE_MIN`, `$CONTROL_PAR_RANGE_MAX`

Improved Features

- `$CONTROL_PAR_BASEPATH` can be set from anywhere in the script, updating the file selector even if the path did not change

KONTAKT 6.5.0

New Features

- New command for checking the bit-mark of an event: `get_event_mark()`
- New built-in variable for checking if KONTAKT is loaded without GUI engine: `$NI_KONTAKT_IS_HEADLESS`
- Added engine parameters for many existing effects:
 - Solid Bus Comp: Link button (`$ENGINE_PAR_SCOMP_LINK`)
 - Classic Compressor: Stereo Link button (`$ENGINE_PAR_COMP_LINK`) and compressor type menu (`$ENGINE_PAR_COMP_TYPE -> $NI_COMP_TYPE_CLASSIC, $NI_COMP_TYPE_ENHANCED, $NI_COMP_TYPE_PRO`)
 - Stereo Modeller: Pseudo Stereo button (`$ENGINE_PAR_STEREO_PSEUDO`)
 - Convolution: Reverse button (`$ENGINE_PAR_IRC_REVERSE`), early reflection/late reflection divider (`$ENGINE_PAR_IRC_ER_LR_BOUNDARY`)
 - Group FX: Post Amp FX slider (`$ENGINE_PAR_POST_FX_SLOT`)
 - Envelope Follower (`$ENGINE_PAR_ENVF_ATTACK, $ENGINE_PAR_ENVF_RELEASE, $ENGINE_PAR_ENVF_GAIN_BOOST, $ENGINE_PAR_ENVF_ADAPTION`)
 - Various others (`$ENGINE_PAR_ENV_AHD, $ENGINE_PAR_ENV_DBD_EASY, $ENGINE_PAR_LFO_NORMALIZE`)

Improved Features

- The maximum number of UI controls has been increased to 999 per UI control type (except for the UI file selector).

KONTAKT 6.4.0

New Features

- New Main Effects signal processing module.
- New engine parameters for new Supercharger GT and Transparent Limiter effects.
- New constants for the **<generic>** argument when setting and getting engine parameters:
\$NI_SEND_BUS, \$NI_INSERT_BUS, \$NI_MAIN_BUS
- New constant that defines which area should be used when dragging from a specific label:
\$CONTROL_PAR_MIDI_EXPORT_AREA_IDX
- New command that defines the number of MIDI object export areas:
mf_set_num_export_areas(<num_of_areas>)
- New command to manage the usage of the new additional export areas:
mf_copy_export_area(<index>)
- New engine parameters for Inverter and Amplifier parameters for Phase Invert and L/R swap:
\$ENGINE_PAR_PHASE_INVERT, \$ENGINE_PAR_LR_SWAP
- New constant allows up to 16 custom event parameters to be assigned:
\$EVENT_PAR_CUSTOM

Improved Features

- The number of maximum MIDI object export areas has been increased to 512.

KONTAKT 6.3.0

New Features

- New constant for handling release velocity: `$EVENT_PAR_REL_VELOCITY`
- New constant for hiding the value display of ui_table: `$HIDE_PART_VALUE`

KONTAKT 6.2.0

New Features

- New Choral, Flair and Phasis modulation effects.
- New UI element: **ui_mouse_area**
- New type of zones accessible from KSP: **set_num_user_zones()**, **set_sample()**, **set_zone_par()**, **set_loop_par()**
- All zone parameters can now be read from KSP: **get_sample()**, **get_zone_par()**, **get_loop_par()**
- New function to check whether a sample is loaded for a zone: **is_zone_empty()**
- New MIR functions to detect zones' pitch, RMS, peak level and loudness.
- New MIR functions to classify samples based on their audio characteristics.
- New command to make handling asynchronous operations more convenient: **wait_async()**

Improved Features

- **purge_group()** now returns an **asyncID**, allowing for reliable tracking of the operations completion.

KONTAKT 6.1.0

New Features

- New engine parameter for the retrigger button on internal modulators (`$ENGINE_PAR_INTMOD_RETRIGGER`)
- New waveform visualization modes (`$CONTROL_PAR_WF_VIS_MODE` with `$NI_WF_VIS_MODE_1`, `$NI_WF_VIS_MODE_2` and `$NI_WF_VIS_MODE_3` as values)
- New Wavetable Mode (`$ENGINE_PAR_WT_INHARMONIC_MODE`)
- New UI Control (`ui_panel`) and related control parameter (`$CONTROL_PAR_PARENT_PANEL`)
- New user interface command (`load_performance_view()`) to load performance views created on Creator Tools

KONTAKT 6.0.2

New Features

- New **engine_par** constants for new KONTAKT 6 effects.
- New **engine_par** constants for new Wavetable mode.
- New UI control: **ui_wavetable** including new commands and built-in variables.
- New commands for variable watching through Creator Tools: **watch_var()** and **watch_array_idx()**
- New control parameter allows deactivating text position shifts when clicking on buttons and switches: **\$CONTROL_PAR_DISABLE_TEXT_SHIFTING**
- New command enables use of custom dynamic fonts: **get_font_id()**
- New control parameters allow granular control over font types for a button's or menu's different states: **\$CONTROL_PAR_FONT_TYPE_ON**, **\$CONTROL_PAR_FONT_TYPE_OFF_PRESSED**, **\$CONTROL_PAR_FONT_TYPE_ON_PRESSED**, **\$CONTROL_PAR_FONT_TYPE_OFF_HOVER** and **\$CONTROL_PAR_FONT_TYPE_ON_HOVER**
- New command allows for quickly disabling emission of messages, warnings or watched variable events to both the KONTAKT Status Bar and Creator Tools: **disable_logging()** with one of the following as the: **\$NI_LOG_MESSAGE**, **\$NI_LOG_WARNING**, **\$NI_LOG_WATCHING**

Improved Features

- New built-in variable and related built-in constants for the XY Pad allow identification of the mouse events that trigger its callback: **\$NI_MOUSE_EVENT_TYPE**, **\$NI_MOUSE_EVENT_TYPE_LEFT_BUTTON_DOWN**, **\$NI_MOUSE_EVENT_TYPE_LEFT_BUTTON_UP** and **\$NI_MOUSE_EVENT_TYPE_DRAG**
- **\$CONTROL_PAR_TEXTPOS_Y** is now allowed on value edit controls.

KONTAKT 5.8.0

Improved Features

- It is now possible to have up to three file selectors per script slot.
- The maximum number of controls per type has now been raised to 512.
- The maximum size for an array has now been raised to 1000000.

KONTAKT 5.7.0

New Features

- New built-in variable for all UI elements: `$CONTROL_PAR_Z_LAYER`
- Waveform styling options: `$CONTROL_PAR_WAVE_COLOR`, `$CONTROL_PAR_BG_COLOR`, `$CONTROL_PAR_WAVE_CURSOR_COLOR`, `$CONTROL_PAR_SLICEMARKERS_COLOR`, `$CONTROL_PAR_BG_ALPHA`
- Engine parameter variables for new effects: ACBox, Cat, DStortion, HotSolo, Van51.
- Added engine parameter variables for effect parameters that are buttons.
- Added engine parameter variables for setting the subtype for the Distortion and Saturator effects: `$ENGINE_PAR_DISTORTION_TYPE`, `$ENGINE_PAR_SHAPE_TYPE`

Improved Features

- `ui_waveform` now accepts `$HIDE_PART_BG` as a `hide_part()` and `$CONTROL_PAR_HIDE` constant.

KONTAKT 5.6.8

New Features

- New built-in UI variables: `$NI_CONTROL_PAR_IDX`, `$HIDE_PART_CURSOR`

KONTAKT 5.6.5

New Features

- New UI control: `ui_xy`
Including new built-in variables: `$CONTROL_PAR_CURSOR_PICTURE`,
`$CONTROL_PAR_MOUSE_MODE`, `$CONTROL_PAR_ACTIVE_INDEX`,
`$CONTROL_PAR_MOUSE_BEHAVIOUR_X`, `$CONTROL_PAR_MOUSE_BEHAVIOUR_Y`
- New UI commands: `set_control_par_arr()` and `set_control_par_str_arr()`

KONTAKT 5.6.0

New Features

- Support for real numbers, including new `~realVariable` and `?realArray[]` types.
- Additional mathematical commands for real numbers.
- New constants: `~NI_MATH_PI` and `~NI_MATH_E`
- New UI commands: `set_ui_color()` and `set_ui_width_px()`
- New control parameter for setting automation IDs via KSP:
`$CONTROL_PAR_AUTOMATION_ID`

KONTAKT 5.5.0

New Features

- New engine parameter variables and built-in constants for controlling the unit parameter of time-related parameters, e.g., `$ENGINE_PAR_DL_TIME_UNIT`, `$NI_SYNC_UNIT_8TH`
- Possible to change FX from KSP by using engine parameter variables for effect type, e.g. `set_engine_par($ENGINE_PAR_EFFECT_TYPE, $EFFECT_TYPE_FILTER, 0, 0, -1)`. For more information, refer to [Changing FX from KSP](#).
- Possible to set Time Machine Pro voice settings: `set_voice_limit()`, `get_voice_limit()`, `$NI_VL_TMPRO_STANDARD`, `$NI_VL_TMRPO_HQ`

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 resource container, additional mode for `load_array()`
- Usage of `load_array_str()` in other callbacks.

KONTAKT 5.3.0

New Features

- Added Engine Parameter Variables for the new Simple Filter effect.

KONTAKT 5.2.0

Improved Features

- Updated 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 5.1.0

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 including many 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`
Including 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`
Including 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`
Including 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 4.2.0

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 4.0.2

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 4.1.0

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 respective zone in the wave editor

Manual Corrections

- Correct Modulation Engine Parameter Variables

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: `$HIDE_PART_NOTHING` and `$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 3.5.0

New Features

- Retrieve the status of a particular event: `event_status()`
- Hide specific parts of UI controls: `hide_part()`

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`
- Addressing modulators by name: `find_mod()` and `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 2.2.0

New Features

- New callback type: `on ui_update`
- New built-in variables for group-based scripting: `$REF_GROUP_IDX` and `%GROUPS_SELECTED`
- Ability to create custom group start options: `NO_SYS_SCRIPT_GROUP_START` (+ various Group Start Options Variables).
- Retrieving the release trigger state of a group: `$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 2.1.0

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 $RPN_VALUE`
`msb()` and `lsb()`
`set_rpn()` and `set_nrpn()`
- Event parameters: `set_event_par()`
- New built-in variables:
`$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.