
KONTAKT 8

Table of Contents

1. Disclaimer

Document authored by Holger Zwar, Mario Krušelj, Yaron Eshkar, Jonas Körwer

Software version: 8.0 (9/2024)

2. Welcome to the Kontakt Lua API Reference Manual

Welcome to the Kontakt Lua API reference manual. Kontakt Lua API is the technology which enables programmatic Kontakt instrument editing and creation workflow through a straightforward imperative approach, directly running within Kontakt itself.

This resource is a reference manual that covers every function, command, constant and other elements of the Kontakt Lua API. Where applicable, it also includes examples and short code snippets that demonstrate how a given function can be used. To learn more about Kontakt, refer to the Kontakt User Manual.

You can navigate this manual through the menu on the left, which groups elements of the API language in functions. Alternatively, if you are looking for a specific command you can use the search bar.

We hope you enjoy exploring the Kontakt Lua API!

Kontakt 8

New Features

- Added `factory_snapshot_path` and `user_snapshot_path` entries to the Instrument Options table.

Fixes

- `set_group_start_options()` did not respect the given group index, it was always applied to all selected groups instead.

3. Introduction and Setup

Document Conventions

Constants and functions in this reference document are described using the following conventions:

```
constant_name: constant_type

function_name(argument_name: argument_type, ...) -> return_type
```

Optional arguments or return values are marked by a question mark `?`. If a table can be passed as an argument, or is returned by a function, its key-value pairs are described below the function.

Basic Usage

Kontakt Lua API scripts can be executed either from Kontakt's user interface, or passed as a command line argument when running Kontakt from a command line.

In Kontakt's main menu the entry “Run Lua script…” (`F11`) opens a file select dialog. In order to run the same script again, press `Ctrl+F11` (`Cmd+F11` on macOS).

Lua scripts also show up in Kontakt's Files browser and can be executed by a double-click or by dragging and dropping a script onto the rack. Drag and drop is also possible from operating system’s file manager (Finder on macOS or Explorer on Windows).

Kontakt Lua API is designed to cooperate with Kontakt's rack and edit views. Running a script operates on the current state of Kontakt, without resetting it. This allows for utility scripts which can do specific tasks that can be cumbersome to do manually in the edit view. If you need to start with a clean state, simply call `Kontakt.reset_multi()`. The constants edit_instrument provide necessary information to retrieve the current context.

In order to start using Kontakt Lua API facilities, please enable Developer Options checkbox in Kontakt’s *Options > Developer* pane.

Tutorials

Tutorial scripts demonstrating various aspects of Kontakt Lua API are available here. These tutorials contain workflow examples with ample comments to get you up to speed, as well as test samples to support various mapping operations.

Command Line and Debugging

If a filename with the extension `.lua` is passed as an argument when opening Kontakt from the command line, the script is executed. The script output is directed to the terminal.

- On Windows, a terminal window will open automatically with Kontakt standalone.
- On macOS, a terminal window will only open if Kontakt is started from the package. Right-click on `Kontakt 7.app` and select *Show Package Contents*, then open `Contents/macOS/Kontakt 7`.

On both platforms, messages from Lua scripts will be visible in the terminal panel of the text editor if Kontakt was launched within the editor. KSP messages, warnings, script errors and variable watching will also be routed to this terminal. However, if Creator Tools is running, all messaging from Kontakt will be redirected to its debugger instead.

This is the recommended workflow. Follow the instructions below for your preferred text editor in order to set this up.

VS Code Setup

1. Press `Ctrl+Shift+P` (`Cmd+Shift+P` on macOS) to open the Command Palette.
2. Search for command *Tasks: Open User Tasks*, then choose **Others** from the list.
3. Paste the text below in the newly created file (replacing everything else), then save.
4. Back in VS Code, in the main menu go to *Terminal > Run Task...* and select **Kontakt 7** to open the terminal along with Kontakt.

Note: On Windows, it is necessary to set the default terminal profile to **Command Prompt**. This is done with the command *Terminal: Select Default Profile*. See here.

`tasks.json`:

```
{
  "version": "2.0.0",
  "tasks": [
    {
      "label": "Kontakt 7",
      "type": "shell",
      "command": "\"/Applications/Native Instruments/Kontakt 7/Kontakt 7.app/
Contents/macOS/Kontakt 7\"",
      "windows": {
        "command": "\"C:\\Program Files\\Native Instruments\\Kontakt 7\\Kontakt
7.exe\"",
      },
      "args": [],
      "problemMatcher": [],
      "presentation": {
        "echo": true,
        "reveal": "always",
        "focus": false,
        "panel": "shared",
        "showReuseMessage": true,
        "clear": true
      },
    }
  ]
}
```

Sublime Text Setup

1. Navigate to *Tools > Build System > New Build System...*

2. Paste the text below in the newly opened file.
3. Save this file as `Kontakt.sublime-build`.
4. Navigate to *Tools > Build System > Kontakt* to execute.

`Kontakt.sublime-build (macOS):`

```
{
  "shell_cmd": ["\"/Applications/Native Instruments/Kontakt 7/Kontakt 7.app/
Contents/macOS/Kontakt 7\"", "$file"],
}
```

`Kontakt.sublime-build (Windows):`

```
{
  "cmd": ["c:\\Program Files\\Native Instruments\\Kontakt 7\\Kontakt 7.exe",
"$file"],
}
```

Note: Sublime Text does not support colored console output. Set colored_output to false in order to disable sending ANSI color codes to output. If the logger is not working, try using `cmd` instead of `shell_cmd`.

Version History

API Access

Kontakt Lua API is defined in the global table `Kontakt`, i.e. all constants and functions need to be prefixed with that name. For example:

```
Kontakt.reset_multi()

-- also works
local kt = Kontakt

kt.reset_multi()
```

Errors

All functions potentially fail and print an error message pointing to the line causing the error. The reason is usually an invalid argument. Consider using *pcall* in order to handle errors.

Modules

It is recommended to move reusable functionality into *Lua modules* and include them via the `require` keyword. Lua's `package.path` is initialized with the folder pointing to the script.

Security

Since potentially unsafe libraries e.g. Operating System Facilities are available, it is highly recommended to run scripts or include modules **from trusted parties only!**

Available Lua Libraries

- Standard libraries are documented in the Lua manual.

4. Multi

General Information

Multi Structure

The multi is the topmost hierarchy level in Kontakt. It consists of **64 instrument slots**. An instrument slot can contain a single insturment, or a single instrument bank, itself consisting of up to **128 instruments** . In Kontakt Lua API, this is represented with **instrument index**, which assumes **128** indices per instrument slot. For example, in order to reference an instrument loaded in third instrument slot, one needs to multiply the 0-based instrument index with 128, which would in this case turns out to be **256**.

There is also a command which retrieves all instrument indices in a table. See get_instrument_indices.

Constants

max_num_multi_scripts

```
max_num_multi_scripts: integer
```

As of Kontakt 7.5, this constant returns **5**.

Get Property

get_multi_name

```
get_multi_name() -> string
```

Returns the name of the multi.

get_multi_script_name

```
get_multi_script_name(multi_script_idx: integer) -> string
```

Returns the title of the multi script in the specified multi script slot.

get_multi_script_source

```
get_multi_script_source(multi_script_idx: integer) -> string
```

Returns the complete multi script contained in the specified multi script slot.

get_num_instruments

```
get_num_instruments() -> integer
```

Returns the number of instruments loaded in the multi.

is_multi_script_bypassed

```
is_multi_script_bypassed(multi_script_idx: integer) -> bool
```

Returns **true** if the specified multi script slot is bypassed.

is_multi_script_protected

```
is_multi_script_protected(multi_script_idx: integer) -> bool
```

Returns **true** if the specified multi script slot is protected with a password.

Set Property

set_multi_name

```
set_multi_name(name: string)
```

Sets the name of the multi.

set_multi_script_bypassed

```
set_multi_script_bypassed(script_idx: integer, bypass: bool)
```

Bypasses the specified script.

set_multi_script_name

```
set_multi_script_name(script_idx: integer, name: string)
```

Sets the name of the script.This will be displayed in the script slot header or the script tab if visible in performance view.

set_multi_script_source

```
set_multi_script_source(script_idx: integer, source: string)
```

Sets the multi script source by passing an absolute filepath as argument.

Modifiers

reset_multi

```
reset_multi()
```

Resets the whole multi to the default state.

File I/O

save_multi

```
save_multi(filename: string, options: table)
```

Save options can be defined by passing a table with one or more of the following entries:

- `mode`: save_modes (default: **"patch"**)
- `absolute_paths`: boolean (default: **false**)
- `compress_samples`: boolean (default: **false**)
- `samples_sub_dir`: string

Individual entries of this table can be omitted. In that case the default value is used.

load_multi

```
load_multi(filename: string)
```

Loads the specified multi. The whole instrument rack is reset beforehand.

5. Instrument

General Information

Instrument Structure

An instrument in Kontakt contains of groups and zones as its basic building blocks. Contrary to Creator Tools, in Kontakt Lua API zones are not hierarchically tied to groups. Instead, they are accessed individually and directly by their own separate index. A zone contains a property which links it to one and only one group. See get_zone_group.

Additionally, an instrument in Kontakt also contains group triggering criteria, voice groups for managing polyphony, playback mode, MIDI input and audio output overrides, polyphonic effects and modulation.

Constants

cc64_modes

```
cc64_modes: table
```

- “midi_exclusive”
- “pedal_and_cc”
- “pedal_exclusive”

instrument_purge_modes

```
instrument_purge_modes: table
```

- “all_samples”
- “reload_all_samples”
- “reset_markers”
- “update_sample_pool”

max_num_instrument_aux

```
max_num_instrument_aux: integer
```

As of Kontakt 7.5, this constant returns **4**.

max_num_instruments

```
max_num_instruments: integer
```

As of Kontakt 7.5, this constant returns **64**.

max_num_scripts

```
max_num_scripts: integer
```

As of Kontakt 7.5, this constant returns **5**.

max_num_voice_groups

```
max_num_voice_groups: integer
```

As of Kontakt 7.5, this constant returns **128**.

save_modes

```
save_modes: table
```

- “monolith”
- “patch”
- “samples”

voice_stealing_modes

```
voice_stealing_modes: table
```

- “any”
- “highest”
- “lowest”

- “newest”
- “oldest”

Get Property

get_free_instrument_index

```
get_free_instrument_index() -> integer?
```

Returns the next available instrument index (not occupied with an already loaded instrument).

get_instrument_aux_level

```
get_instrument_aux_level(instrument_idx: integer, aux_index: integer) -> float
```

Returns the specified aux send level of the specified instrument.

get_instrument_indices

```
get_instrument_indices() -> table
```

The returned table contains the indices of all instruments. Pass these indices to functions taking `instrument_idx` as an argument.

get_instrument_midi_channel

```
get_instrument_midi_channel(instrument_idx: integer) -> integer
```

Returns the MIDI channel of the specified instrument.

get_instrument_mute

```
get_instrument_mute(instrument_idx: integer) -> bool
```

Returns the mute state of the specified instrument.

get_instrument_name

```
get_instrument_name(instrument_idx: integer) -> string
```

Returns the name of the specified instrument.

get_instrument_options

```
get_instrument_options(instrument_idx: integer) -> table
```

Returns specified instrument's options as a table with the following entries:

Instrument

- `key_switch`: integer (default: **nil**)
- `key_range_from`: integer (default: **0**)
- `key_range_to`: integer (default: **127**)
- `velocity_range_from`: integer (default: **0**)
- `velocity_range_to`: integer (default: **127**)
- `midi_transpose`: integer (default: **0**)
- `wallpaper`: string (default: **nil**)

Voice Handling

- `voice_stealing_mode`: voice_stealing_modes (default: **"oldest"**)

- `voice_stealing_fadeout`: integer (default: **10**)
- `time_machine_voice_limit`: integer (default: **8**)
- `time_machine_voice_limit_hq`: integer (default: **4**)
- `time_machine_use_legacy`: boolean (default: **false**)

DFD

- `dfd_buffersize`: integer (default: **60**)
- `background_loading`: boolean (default: **true**)

Controller

- `cc_64_mode`: cc64_modes (default: **"pedal_and_cc"**)
- `use_cc_120_123`: boolean (default: **true**)
- `use_cc_7_10`: boolean (default: **true**)
- `cc_7_range`: integer (default: **0**)

Snapshots

- `show_factory_snapshots`: boolean (default: **true**)
- `factory_snapshot_path`: string (default: depends on instrument, or **nil**)
- `user_snapshot_path`: string (default: depends on instrument, or **nil**)

Info

- `info_icon`: integer (default: **28**)
- `info`: string (default: **"(null)"**)
- `info_author`: string (default: **"Kontakt"**)
- `info_url`: string (default: **"(null)"**)

Individual entries of this table can be omitted. In that case the default value is used.

get_instrument_output_channel

```
get_instrument_output_channel(instrument_idx: integer) -> integer
```

Returns the audio output channel of the specified instrument.

get_instrument_pan

```
get_instrument_pan(instrument_idx: integer) -> float
```

Returns the output panorama of the specified instrument.

get_instrument_polyphony

```
get_instrument_polyphony(instrument_idx: integer) -> integer
```

Returns the maximum polyphony of the specified instrument.

get_instrument_script_linked_filename

```
get_instrument_script_linked_filename(instrument_idx: integer, script_idx: integer)
-> string
```

Returns the filename of the script linked to the specified script slot in the specified instrument.

get_instrument_script_name

```
get_instrument_script_name(instrument_idx: integer, script_idx: integer) -> string
```

Returns the title of the script in the specified script slot in the specified instrument.

get_instrument_script_source

```
get_instrument_script_source(instrument_idx: integer, script_idx: integer) -> string
```

Returns the complete script contained in the specified script slot in the specified instrument.

get_instrument_solo

```
get_instrument_solo(instrument_idx: integer) -> bool
```

Returns the solo state of the specified instrument.

get_instrument_tune

```
get_instrument_tune(instrument_idx: integer) -> float
```

Returns the tuning of the specified instrument in semitones.

get_instrument_volume

```
get_instrument_volume(instrument_idx: integer) -> float
```

Returns the output volume of the specified instrument in dB.

get_voice_groups

```
get_voice_groups(instrument_idx: integer) -> table
```

Returns all voice groups of the specified instrument as a table with a maximum of 128 entries.

is_instrument_script_bypassed

```
is_instrument_script_bypassed(instrument_idx: integer, script_idx: integer) -> bool
```

Returns **true** if the specified script slot in the specified instrument is bypassed.

is_instrument_script_linked

```
is_instrument_script_linked(instrument_idx: integer, script_idx: integer) -> bool
```

Returns **true** if the script at the specified script slot of the specified instrument index is linked to a file in the resource container.

is_instrument_script_protected

```
is_instrument_script_protected(instrument_idx: integer, script_idx: integer) -> bool
```

Returns **true** if the specified script slot in the specified instrument is protected with a password.

Set Property

reset_instrument

```
reset_instrument(instrument_idx: integer)
```

Resets the whole instrument at the specified instrument index to the default state.

set_instrument_aux_level

```
set_instrument_aux_level(instrument_idx: integer, aux_index: integer, level: float)
```

Sets the level of the specified aux index for the specified instrument in dB. Range is **-math.huge … 12.0**.

set_instrument_midi_channel

```
set_instrument_midi_channel(instrument_idx: integer, channel: integer)
```

Sets the MIDI channel for the specified instrument. **0** is Omni, **1 … 64** is channels 1-16 across ports A-D.

set_instrument_mute

```
set_instrument_mute(instrument_idx: integer, value: bool)
```

Mutes the specified instrument.

set_instrument_name

```
set_instrument_name(instrument_idx: integer, name: string)
```

Sets the name of the specified instrument.

set_instrument_options

```
set_instrument_options(instrument_idx: integer, options: table)
```

Specified instrument's options can be set by passing a table with one or more of the following entries:

Instrument

- `key_switch`: integer (default: **nil**)
- `key_range_from`: integer (default: **0**)
- `key_range_to`: integer (default: **127**)
- `velocity_range_from`: integer (default: **0**)
- `velocity_range_to`: integer (default: **127**)
- `midi_transpose`: integer (default: **0**)
- `wallpaper`: string (default: **nil**)

Voice Handling

- `voice_stealing_mode`: voice_stealing_modes (default: **"oldest"**)
- `voice_stealing_fadeout`: integer (default: **10**)
- `time_machine_voice_limit`: integer (default: **8**)

- `time_machine_voice_limit_hq`: integer (default: **4**)
- `time_machine_use_legacy`: boolean (default: **false**)

DFD

- `dfd_buffersize`: integer (default: **60**)
- `background_loading`: boolean (default: **true**)

Controller

- `cc_64_mode`: cc64_modes (default: **"pedal_and_cc"**)
- `use_cc_120_123`: boolean (default: **true**)
- `use_cc_7_10`: boolean (default: **true**)
- `cc_7_range`: integer (default: **0**)

Snapshots

- `show_factory_snapshots`: boolean (default: **true**)
- `factory_snapshot_path`: string (default: depends on instrument, or **nil**)
- `user_snapshot_path`: string (default: depends on instrument, or **nil**)

Info

- `info_icon`: integer (default: **28**)
- `info`: string (default: **"(null)"**)
- `info_author`: string (default: **"Kontakt"**)
- `info_url`: string (default: **"(null)"**)

Individual entries of this table can be omitted. In that case the default value is used.

set_instrument_output_channel

```
set_instrument_output_channel(instrument_idx: integer, channel: integer)
```

Sets the audio output for the specified instrument. Make sure to check how many outputs are available before setting!

set_instrument_pan

```
set_instrument_pan(instrument_idx: integer, percent: float)
```

Sets the output panorama of the specified instrument. Range is **-100.0 … 100.0**.

set_instrument_polyphony

```
set_instrument_polyphony(instrument_idx: integer, voices: integer)
```

Sets the maximum polyphony for the specified instrument. Minimum value is **1**.

set_instrument_script_bypassed

```
set_instrument_script_bypassed(instrument_idx: integer, script_idx: integer,
bypass: bool)
```

Bypasses the specified script of the specified instrument.

set_instrument_script_linked_filename

```
set_instrument_script_linked_filename(instrument_idx: integer, script_idx: integer, filename: string)
```

Sets the filename of the script packed inside the resource container and linked to a script slot.

set_instrument_script_name

```
set_instrument_script_name(instrument_idx: integer, script_idx: integer, name: string)
```

Sets the name of the script (without extension). This will be displayed in the script slot header or the script tab if visible in performance view.

set_instrument_script_source

```
set_instrument_script_source(instrument_idx: integer, script_idx: integer, source: string)
```

Sets the script source by passing an absolute filepath as argument.

set_instrument_solo

```
set_instrument_solo(instrument_idx: integer, value: bool)
```

Soloes the specified instrument.

set_instrument_tune

```
set_instrument_tune(instrument_idx: integer, semitones: float)
```

Sets the output tuning of the specified instrument in semitones. Range is **-36.0 … 36.0**.

set_instrument_volume

```
set_instrument_volume(instrument_idx: integer, level: float)
```

Sets the output volume of the specified instrument in dB. Range is **-math.huge … 12.0**.

set_voice_groups

```
set_voice_groups(instrument_idx: integer, voice_groups: table)
```

Sets voice groups of the specified instrument as a table with a maximum of 128 entries, matching the total number of possible voice groups. Nil table entries will set voice group to default values. Voice group parameters are defined in a sub-table:

- `mode`: voice_group_modes (default: **"oldest"**)
- `name`: string (default: '')
- `voices`: integer (default: **1**)
- `fade_time`: integer (default: **10**)
- `prefer_released`: boolean (default: **true**)
- `exclusive_group`: integer (default: **nil**)

Individual entries of this table can be omitted. In that case the default value is used.

Modifiers

add_instrument

```
add_instrument(instrument_idx: integer?) -> integer
```

Inserts a new instrument at given or next available instrument index. Returns the index of the new instrument. Pass this index to functions taking `instrument_idx` as an argument.

add_instrument_bank

```
add_instrument_bank(instrument_slot: integer?) -> integer
```

Inserts a new instrument bank at given or next available instrument slot. Returns the instrument slot of the new instrument bank.

remove_instrument

```
remove_instrument(instrument_idx: integer)
```

Removes the instrument at the specified instrument index from the multi.

remove_instrument_bank

```
remove_instrument_bank(instrument_slot: integer)
```

Removes the instrument bank at the specified instrument slot from the multi.

File I/O

load_instrument

```
load_instrument(filename: string, instrument_idx: integer?) -> integer
```

Loads an instrument to the specified slot index. If that slot is already occupied, next available slot is used. Returns the slot index of the new instrument. **Note:** contrary to most other functions, the slot index here can also refer to a slot within an instrument bank!

load_snapshot

```
load_snapshot(instrument_idx: integer, filename: string)
```

Loads the specified snapshot in the specified instrument index.

save_instrument

```
save_instrument(instrument_idx: integer, filename: string, options: table)
```

Save options can be defined by passing a table with one or more of the following entries:

- `mode`: save_modes (default: **"patch"**)
- `absolute_paths`: boolean (default: **false**)
- `compress_samples`: boolean (default: **false**)
- `samples_sub_dir`: string

Individual entries of this table can be omitted. In that case the default value is used.

save_snapshot

```
save_snapshot(instrument_idx: integer, filename: string)
```

Saves the state of the instrument at the specified insturment index as a snapshot at the specified absolute path.

6. Group

Constants

group_playback_modes

```
group_playback_modes: table
```

- “beat_machine”
- “dfd”
- “mpc60_machine”
- “s1200_machine”
- “sampler”
- “time_machine_1”
- “time_machine_2”
- “time_machine_pro”
- “tone_machine”
- “wavetable”

group_start_conditions

```
group_start_conditions: table
```

- “controller”
- “key”
- “random”
- “round_robin”
- “slice_trigger”

group_start_operators

```
group_start_operators: table
```

- “and”
- “and_not”
- “or”

max_num_group_start_criteria

```
max_num_group_start_criteria: integer
```

As of Kontakt 7.5, this constant returns **4**.

max_num_groups

```
max_num_groups: integer
```

As of Kontakt 7.5, this constant returns **4096**.

voice_group_modes

```
voice_group_modes: table
```

- “any”

- “highest”
- “lowest”
- “newest”
- “oldest”

Get Property

get_group_name

```
get_group_name(instrument_idx: integer, group_idx: integer) -> string
```

Returns the name of the specified group.

get_group_pan

```
get_group_pan(instrument_idx: integer, group_idx: integer) -> float
```

Returns the amplifier panorama of the specified group.

get_group_playback_mode

```
get_group_playback_mode(instrument_idx: integer, group_idx: integer) -> string
```

Returns the playback mode of the specified group's Source module. See group_playback_modes.

get_group_start_options

```
get_group_start_options(instrument_idx: integer, group_idx: integer) -> table
```

Returns specified group's start options as a table with a maximum of 4 sub-tables:

A certain group start option condition can be defined by passing a table. This table has different entries based on condition type:

Start On Key

- `mode`: "key"
- `key_min`: integer (default: **24**)
- `key_max`: integer (default: **24**)
- `next`: group_start_operators (default: **"and"**)

Start On Controller

- `mode`: "controller"
- `controller`: (default: **1**)
- `cc_min`: (default: **0**)
- `cc_max`: (default: **64**)
- `next`: group_start_operators (default: **"and"**)

Cycle Round Robin

- `mode`: "round_robin"
- `position`: integer (default: **1**)
- `next`: group_start_operators (default: **"and"**)

Cycle Random

- `mode`: "random"
- `next`: group_start_operators (default: **"and"**)

Slice Trigger

- `mode`: "slice_trigger"
- `zone`: integer (default: **nil**)

- `slice`: integer (default: **nil**)
- `internal`: boolean (default: **false**)
- `next`: group_start_operators (default: **"and"**)

Individual entries of this table can be omitted. In that case the default value is used. See also: group_start_conditions

get_group_tune

```
get_group_tune(instrument_idx: integer, group_idx: integer) -> float
```

Returns the tuning of the specified group in semitones.

get_group_volume

```
get_group_volume(instrument_idx: integer, group_idx: integer) -> float
```

Returns the amplifier volume of the specified group in dB.

get_num_groups

```
get_num_groups(instrument_idx: integer) -> integer
```

Returns the total number of groups in the specified instrument.

get_voice_group

```
get_voice_group(instrument_idx: integer, group_idx: integer) -> integer?
```

Returns the voice group assigned to a group. If no voice group is assigned, the returned value is **nil**.

Set Property

set_group_name

```
set_group_name(instrument_idx: integer, group_idx: integer, name: string)
```

Sets the name of the specified group.

set_group_pan

```
set_group_pan(instrument_idx: integer, group_idx: integer, pan: float)
```

Sets the amplifier panorama of the specified group. Range is **-100.0 … 100.0**.

set_group_playback_mode

```
set_group_playback_mode(instrument_idx: integer, group_idx: integer, mode: string)
```

Sets the playback mode of the specified group's Source module. See group_playback_modes.

set_group_start_options

```
set_group_start_options(instrument_idx: integer, group_idx: integer, options: table)
```

Start On Key

- `mode`: "key"
- `key_min`: integer (default: **24**)
- `key_max`: integer (default: **24**)
- `next`: group_start_operators (default: **"and"**)

Start On Controller

- `mode`: "controller"
- `controller`: (default: **1**)
- `cc_min`: (default: **0**)
- `cc_max`: (default: **64**)
- `next`: group_start_operators (default: **"and"**)

Cycle Round Robin

- `mode`: "round_robin"
- `position`: integer (default: **1**)
- `next`: group_start_operators (default: **"and"**)

Cycle Random

- `mode`: "random"
- `next`: group_start_operators (default: **"and"**)

Slice Trigger

- `mode`: "slice_trigger"
- `zone`: integer (default: **nil**)
- `slice`: integer (default: **nil**)
- `internal`: boolean (default: **false**)
- `next`: group_start_operators (default: **"and"**)

Individual entries of this table can be omitted. In that case the default value is used. See also: group_start_conditions

set_group_tune

```
set_group_tune(instrument_idx: integer, group_idx: integer, tune: float)
```

Sets the tuning of the specified group in semitones. Range is **-36.0 … 36.0**.

set_group_volume

```
set_group_volume(instrument_idx: integer, group_idx: integer, volume: float)
```

Sets the amplifier volume of the specified group in dB. Range is **-math.huge … 12.0**.

set_voice_group

```
set_voice_group(instrument_idx: integer, group_idx: integer, voice_group: integer)
```

Assign a voice group to a group. In order to reset the assignment pass **nil**.

Modifiers

add_group

```
add_group(instrument_idx: integer) -> integer
```

Returns the index of the new group. Pass this index to functions taking `group_idx` as an argument.

remove_group

```
remove_group(instrument_idx: integer, group_idx: integer) -> bool
```

Removes the specified group from the specified instrument.

File I/O

load_group

```
load_group(instrument_idx: integer, group_idx: integer, filename: string, options: table)
```

Options table currently has a single entry:

- `replace_zones`: boolean (default: **false**)

save_group

```
save_group(instrument_idx: integer, group_idx: integer, filename: string, options: table)
```

Save options can be defined by passing a table with one or more of the following entries:

- `mode`: save_modes (default: **"patch"**)
- `absolute_paths`: boolean (default: **false**)
- `compress_samples`: boolean (default: **false**)
- `samples_sub_dir`: string

Individual entries of this table can be omitted. In that case the default value is used.

7. Zone

Constants

max_num_sample_loops

```
max_num_sample_loops: integer
```

As of Kontakt 7.5, this constant returns **8**.

max_num_zones

```
max_num_zones: integer
```

As of Kontakt 7.5, this constant returns **98304**.

sample_loop_modes

```
sample_loop_modes: table
```

- “off”
- “until_end”
- “until_end_alt”
- “until_release”
- “until_release_alt”

zone_grid_modes

```
zone_grid_modes: table
```

- “auto”
- “fixed”
- “none”

Get Property

get_num_zones

```
get_num_zones(instrument_idx: integer) -> integer
```

Returns the total number of zones in the specified instrument.

get_sample_loop_count

```
get_sample_loop_count(instrument_idx: integer, zone_idx: integer, loop_idx:
integer) -> integer
```

Returns the loop count of the specified zone's loop.

get_sample_loop_length

```
get_sample_loop_length(instrument_idx: integer, zone_idx: integer, loop_idx:
integer) -> integer
```

Returns the loop length of the specified zone's loop.

get_sample_loop_mode

```
get_sample_loop_mode(instrument_idx: integer, zone_idx: integer, loop_idx: integer)
-> string
```

Returns the loop mode of the specified zone's loop. See sample_loop_modes.

get_sample_loop_start

```
get_sample_loop_start(instrument_idx: integer, zone_idx: integer, loop_idx:
integer) -> integer
```

Returns the loop start of the specified zone's loop.

get_sample_loop_tune

```
get_sample_loop_tune(instrument_idx: integer, zone_idx: integer, loop_idx: integer)
-> float
```

Returns the loop tuning of the specified zone's loop in semitones.

get_sample_loop_xfade

```
get_sample_loop_xfade(instrument_idx: integer, zone_idx: integer, loop_idx:
integer) -> integer
```

Returns the loop crossfade time of the specified zone's loop.

get_zone_geometry

```
get_zone_geometry(instrument_idx: integer, zone_idx: integer) -> table
```

Returns a table containing the complete zone geometry (key and velocity ranges, root key, zone crossfades, etc.). See set_zone_geometry.

get_zone_grid_bpm

```
get_zone_grid_bpm(instrument_idx: integer, zone_idx: integer) -> float
```

Returns the BPM of the specified zone.

get_zone_grid_mode

```
get_zone_grid_mode(instrument_idx: integer, zone_idx: integer) -> string
```

Returns the grid mode of the specified zone. See zone_grid_modes.

get_zone_group

```
get_zone_group(instrument_idx: integer, zone_idx: integer) -> integer
```

Returns the index of the group which contains the specified zone.

get_zone_high_key

```
get_zone_high_key(instrument_idx: integer, zone_idx: integer) -> integer
```

Returns the high key of the specified zone.

get_zone_high_key_fade

```
get_zone_high_key_fade(instrument_idx: integer, zone_idx: integer) -> integer
```

Returns the high key crossfade span of the specified zone.

get_zone_high_velocity

```
get_zone_high_velocity(instrument_idx: integer, zone_idx: integer) -> integer
```

Returns the high velocity of the specified zone.

get_zone_high_velocity_fade

```
get_zone_high_velocity_fade(instrument_idx: integer, zone_idx: integer) -> integer
```

Returns the high velocity crossfade span of the specified zone.

get_zone_low_key

```
get_zone_low_key(instrument_idx: integer, zone_idx: integer) -> integer
```

Returns the low key of the specified zone.

get_zone_low_key_fade

```
get_zone_low_key_fade(instrument_idx: integer, zone_idx: integer) -> integer
```

Returns the low key crossfade span of the specified zone.

get_zone_low_velocity

```
get_zone_low_velocity(instrument_idx: integer, zone_idx: integer) -> float
```

Returns the low velocity of the specified zone.

get_zone_low_velocity_fade

```
get_zone_low_velocity_fade(instrument_idx: integer, zone_idx: integer) -> float
```

Returns the low velocity crossfade span of the specified zone.

get_zone_pan

```
get_zone_pan(instrument_idx: integer, group_idx: integer) -> float
```

Returns the panorama offset of the specified zone.

get_zone_root_key

```
get_zone_root_key(instrument_idx: integer, zone_idx: integer) -> integer
```

Returns the root key of the specified zone.

get_zone_sample

```
get_zone_sample(instrument_idx: integer, zone_idx: integer) -> string?
```

Returns the absolute file path of the sample loaded in the specified zone.

get_zone_sample_channels

```
get_zone_sample_channels(instrument_idx: integer, zone_idx: integer) -> integer?
```

Returns the number of audio channels in the sample loaded in the specified zone.

get_zone_sample_end

```
get_zone_sample_end(instrument_idx: integer, zone_idx: integer) -> integer
```

Returns the sample end position for the specified zone. This is expressed in negative frames, by how far away the sample end marker is from last sample frame, so in order to get the same value as displayed in Kontakt's Wave Editor, use `get_zone_sample_frames() + get_zone_sample_end()`.

get_zone_sample_frames

```
get_zone_sample_frames(instrument_idx: integer, zone_idx: integer) -> integer?
```

Returns the length of the sample loaded in the specified zone.

get_zone_sample_start

```
get_zone_sample_start(instrument_idx: integer, zone_idx: integer) -> integer
```

Returns the sample start position for the specified zone.

get_zone_sample_start_mod_range

```
get_zone_sample_start_mod_range(instrument_idx: integer, zone_idx: integer) ->
integer
```

Returns the sample start modulation range for the specified zone.

get_zone_tune

```
get_zone_tune(instrument_idx: integer, zone_idx: integer) -> float
```

Returns the tuning offset of the specified zone in semitones.

get_zone_volume

```
get_zone_volume(instrument_idx: integer, zone_idx: integer) -> float
```

Returns the volume offset of the specified zone in dB.

Set Property

set_sample_loop_count

```
set_sample_loop_count(instrument_idx: integer, zone_idx: integer, loop_idx:
integer, count: integer)
```

Sets the loop count of the specified zone's loop.

set_sample_loop_length

```
set_sample_loop_length(instrument_idx: integer, zone_idx: integer, loop_idx:
integer, frames: integer)
```

Sets the length for the specified zone's loop in sample frames.

set_sample_loop_mode

```
set_sample_loop_mode(instrument_idx: integer, zone_idx: integer, loop_idx: integer,
mode: string)
```

Sets the playback mode for the specified zone's loop.See sample_loop_modes.

set_sample_loop_start

```
set_sample_loop_start(instrument_idx: integer, zone_idx: integer, loop_idx:
integer, frame: integer)
```

Sets the start of the specified zone's loop.

set_sample_loop_tune

```
set_sample_loop_tune(instrument_idx: integer, zone_idx: integer, loop_idx: integer,
tune: float)
```

Sets the tuning of the specified zone's loop in semitones. Range is **-12.0 … 12.0**.

set_sample_loop_xfade

```
set_sample_loop_xfade(instrument_idx: integer, zone_idx: integer, loop_idx:
integer, frames: integer)
```

Sets the length of the specified zone's loop xfade in sample frames.

set_zone_geometry

```
set_zone_geometry(instrument_idx: integer, zone_idx: integer, geometry: table)
```

Applies `geometry` to the specified zone. This consists of a table of the following properties which specify the boundaries of the zone.

Geometry:

- `root_key` (default: **36**, range: **0 … 127**)
- `low_key` (default: **0**, range: **0** ... **`high_key`**)
- `high_key` (default: **127**, range: **`low_key`** ...**127**)
- `low_key_fade` (default: **0**, range: **0** ... **span**)
- `high_key_fade` (default: **0**, range: **0** ... **span**)

- `low_velocity` (default: **1**, range: **0** ... **`high_velocity`**)
- `high_velocity` (default: **127**, range: **`low_velocity`** ... **127**)
- `low_velocity_fade` (default: **0**, range: **0** ... **span**)
- `high_velocity_fade` (default: **0**, range: **0** ... **span**)

`span` is defined as: `high_key / velocity + 1 - low_key / velocity - opposite_fade_value`. So if setting `low_key_fade`, use the value of `high_key_fade` as `opposite_fade_value`

set_zone_grid

```
set_zone_grid(instrument_idx: integer, zone_idx: integer, mode: string, bpm: float)
```

Sets the grid of the specified zone. See zone_grid_modes.

set_zone_high_key

```
set_zone_high_key(instrument_idx: integer, zone_idx: integer, key: integer)
```

See set_zone_geometry.

set_zone_high_key_fade

```
set_zone_high_key_fade(instrument_idx: integer, zone_idx: integer, key_fade: integer)
```

See set_zone_geometry.

set_zone_high_velocity

```
set_zone_high_velocity(instrument_idx: integer, zone_idx: integer, velocity: integer)
```

See set_zone_geometry.

set_zone_high_velocity_fade

```
set_zone_high_velocity_fade(instrument_idx: integer, zone_idx: integer, velocity_fade: integer)
```

See set_zone_geometry.

set_zone_low_key

```
set_zone_low_key(instrument_idx: integer, zone_idx: integer, key: integer)
```

See set_zone_geometry.

set_zone_low_key_fade

```
set_zone_low_key_fade(instrument_idx: integer, zone_idx: integer, key_fade: integer)
```

See set_zone_geometry.

set_zone_low_velocity

```
set_zone_low_velocity(instrument_idx: integer, zone_idx: integer, velocity: integer)
```

See set_zone_geometry.

set_zone_low_velocity_fade

```
set_zone_low_velocity_fade(instrument_idx: integer, zone_idx: integer,
velocity_fade: integer)
```

See set_zone_geometry.

set_zone_pan

```
set_zone_pan(instrument_idx: integer, zone_idx: integer, pan: float)
```

Sets the panorama offset of the specified zone. Range is **-100.0 … 100.0**.

set_zone_root_key

```
set_zone_root_key(instrument_idx: integer, zone_idx: integer, key: integer)
```

See set_zone_geometry.

set_zone_sample

```
set_zone_sample(instrument_idx: integer, zone_idx: integer, filename: string)
```

Sets the absolute path of the sample to be loaded in the specified zone.

set_zone_sample_end

```
set_zone_sample_end(instrument_idx: integer, zone_idx: integer, frame: integer)
```

Sets the sample end of the sample loaded in the specified zone. This is expressed in negative frames, by how far away the sample end marker is from last sample frame.

set_zone_sample_start

```
set_zone_sample_start(instrument_idx: integer, zone_idx: integer, frame: integer)
```

Sets the sample start of the sample loaded in the specified zone, in sample frames.

set_zone_sample_start_mod_range

```
set_zone_sample_start_mod_range(instrument_idx: integer, zone_idx: integer, frame:
integer)
```

Sets the sample start modulation range of the sample loaded in the specified zone, in sample frames.

set_zone_tune

```
set_zone_tune(instrument_idx: integer, zone_idx: integer, tune: float)
```

Sets the tuning offset of the specified zone in semitones. Range is **-36.0** … **36.0**.

set_zone_volume

```
set_zone_volume(instrument_idx: integer, zone_idx: integer, volume: float)
```

Sets the volume offset of the specified zone in dB. Range is **-36.0** … **36.0**.

Modifiers

add_zone

```
add_zone(instrument_idx: integer, group_idx: integer, filename: string) -> integer
```

The returned index refers the added zone. Pass this index to functions taking `zone_idx` as an argument.

remove_zone

```
remove_zone(instrument_idx: integer, zone_idx: integer)
```

Removes the specified zone index from the instrument at the specified instrument index.

restore_loops_from_sample

```
restore_loops_from_sample(instrument_idx: integer, zone_idx: integer)
```

Sets the loop points to the values stored in the sample metadata, if it exists.

8. Presets

Constants

bus_fx

```
bus_fx: integer
```

Points a preset loading function to the first instrument bus FX chain. To reach other instrument buses, add a constant from **1** to **15**.

group_fx

```
group_fx: integer
```

Points a preset loading function to the group FX chain.

insert_fx

```
insert_fx: integer
```

Points a preset loading function to the insert FX chain.

main_fx

```
main_fx: integer
```

Points a preset loading function to the main FX chain.

output_fx

```
output_fx: integer
```

Points a preset loading function to the output FX chain. Does not apply to `load_fx_chain_preset()`.

send_fx

```
send_fx: integer
```

Points a preset loading function to the send FX chain.

File I/O

load_source_preset

```
load_source_preset(filename: string, instrument_idx: integer, group_idx: integer)
-> integer
```

Loads a source preset file (.NKP) to the specified group of an instrument.

load_fx_preset

```
load_fx_preset(filename: string, instrument_or_output_idx: integer, group_idx:
integer, generic: integer) -> integer
```

Loads an effect preset file (.NKP) to the specified location based on the following arguments:

- `instrument_or_output`: if `generic` argument is `output_fx`, this is one of output channels in the Output panel (**0 … 127**), else it's an instrument index
- `group`: only valid when `generic` argument is `group_fx`, else set to **-1**
- `slot`: index of the slot to which the effect preset should be loaded
- `generic`: use one of FX chain location constants (`group_fx`, `send_fx`, etc.)

load_multi_script_preset

```
load_multi_script_preset(filename: string, instrument_idx: integer,
multi_script_idx: integer) -> integer
```

Loads a multi script preset file (.NKP) to the specified multi script slot.

load_script_preset

```
load_script_preset(filename: string, instrument_idx: integer, script_idx: integer)
-> integer
```

Loads a script preset file (.NKP) to the specified script slot of an instrument.

load_fx_chain_preset

```
load_fx_chain_preset(filename: string, instrument_idx: integer, group_idx: integer,
generic: integer) -> integer
```

Loads a complete effect chain preset file (.NKP) to the specified location based on the following arguments:

- `instrument`: instrument index
- `group`: only valid when `generic` argument is `group_fx`, else set to **-1**
- `generic`: use one of FX chain location constants, except `output_fx`

9. Options

Constants

desktop_path

```
desktop_path: string
```

Returns the absolute path to operating system's Desktop folder.

documents_path

```
documents_path: string
```

Returns the absolute path to operating system's Documents folder.

factory_path

```
factory_path: string
```

Returns the Kontakt factory data path (contains wavetables, NKP presets, and so on).

macos

```
macos: bool
```

Returns **true** if the system running the Lua script is macOS.

ni_content_path

```
ni_content_path: string
```

Returns the absolute path to NI Content folder.

non_player_content_base_path

```
non_player_content_base_path: string
```

Returns the non-Player content base path (which is optionally set in Kontakt's *Options > Loading* pane).

snapshot_path

```
snapshot_path: string
```

Returns the User Content data path, which contains snapshots saved by the user.

windows

```
windows: bool
```

Returns **true** if the system running the Lua script is Windows.

10. Utility

Constants

colored_output

```
colored_output: bool
```

Controls if terminal output is colored or not.

edit_instrument

```
edit_instrument: integer | nil
```

Index of the currently edited instrument.

script_executed_from_instrument

```
script_executed_from_instrument: integer | nil
```

Index of the instrument slot from which the script was loaded. **nil** is multi.

script_file

```
script_file: string
```

Returns filename of the currently running Lua script, including the extension.

script_path

```
script_path: string
```

Returns the absolute path to the folder containing the currently running Lua script.

version

```
version: string
```

Returns the current version of the running instance of Kontakt.

Functions

assert_fail

```
assert_fail(test: function)
```

Unit test helper function. Calls `test` expecting it to fail. Raises an error if that does not happen.

create_resource_container

```
create_resource_container(instrument_idx: integer, filename: string)
```

Creates a new resource container (with the obligatory Resources folder) or updates an existing resource container. If the .nkr extension is not a part of the filename string, it will be added automatically.

link_resource_container

```
link_resource_container(instrument_idx: integer, filename: string)
```

Links an existing resource container to the instrument. If the .nkr extension is not a part of the filename string, it will be added automatically. Note that this command does not require the Resources folder to exist.

get_file_info

```
get_file_info(filename: string) -> table
```

Extracts information from an instrument on disk. The returned table contains the fields: `file`, `format`, `version`, `library`, `num_instruments`, `num_groups`, `num_zones`.

instrument_purge

```
instrument_purge(instrument_idx: integer, mode: string)
```

Runs one of available purge actions for the specified instrument. See instrument_purge_modes.

File I/O

ncw_decode

```
ncw_decode(source: string, target: string)
```

Decodes an NCW sample back to uncompressed WAV format.

ncw_encode

```
ncw_encode(source: string, target: string)
```

Encodes a WAV or AIFF sample to losslessly compressed NCW format.

11. Filesystem

The Lua binding is based on the C++ library boost filesystem. In contrary to the original C++ design, the Lua binding does not define an abstraction for path. Instead, path always refers to a Lua string. The join function allows concatenating paths using the platform dependent separator.

```
local fs = Filesystem

local script = fs.join(Kontakt.script_path, Kontakt.script_file)
assert(fs.exists(script))
```

API Access

The Filesystem API is defined in the global table `Filesystem`, i.e. all constants and functions need to be prefixed with that name. For example.

```
print(Filesystem.exists(Kontakt.script_path))

-- also works
kt = Kontakt
fs = Filesystem

print(fs.exists(kt.script_path))
```

Iterators

directory

```
directory(path: string) -> iterator
```

Example: lists paths in directory

```
for _, p in Filesystem.directory(path) do
    print(p)
end
```

recursive_directory

```
recursive_directory(path: string) -> iterator
```

Example: lists paths in directory and all sub-directories

```
for _, p in Filesystem.recursive_directory(path) do
    print(p)
end
```

Path Functions

empty

```
empty(path: string) -> boolean
```

extension

```
extension(path: string) -> string
```

filename

```
filename(path: string) -> string
```

has_parent_path

```
has_parent_path(path: string) -> boolean
```

has_relative_path

```
has_relative_path(path: string) -> boolean
```

has_root_directory

```
has_root_directory(path: string) -> boolean
```

has_extension

```
has_extension(path: string) -> boolean
```

has_filename

```
has_filename(path: string) -> boolean
```

has_root_name

```
has_root_name(path: string) -> boolean
```

is_dot_dot

```
is_dot_dot(path: string) -> boolean
```

is_dot

```
is_dot(path: string) -> boolean
```

join

```
join(paths ... : string) -> string
```

has_stem

```
has_stem(path: string) -> boolean
```

is_absolute

```
is_absolute(path: string) -> boolean
```

relative_path

```
relative_path(path: string) -> string
```

has_root_path

```
has_root_path(path: string) -> boolean
```

root_name

```
root_name(path: string) -> string
```

root_directory

```
root_directory(path: string) -> string
```

is_relative

```
is_relative(path: string) -> boolean
```

parent_path

```
parent_path(path: string) -> string
```

root_path

```
root_path(path: string) -> string
```

replace_extension

```
replace_extension(path: string, extension: string) -> string
```

preferred

```
preferred(path: string) -> string
```

stem

```
stem(path: string) -> string
```

12. Music Information Retrieval (MIR)

Music Information Retrieval is the science of retrieving information from music. Among others, it allows the extraction of meaningful features from audio files, such as the pitch or the loudness of a sample. Kontakt comes with a collection of MIR functions, to assist or automate parts of the instrument creation process.

Single functions retrieve information from single files and take as argument an absolute filename (the full path to the sample file).

API Access

The MIR API is defined in the global table `MIR`, i.e. all constants and functions need to be prefixed with that name. For example:

```
kt = Kontakt
fs = Filesystem

print(MIR.detect_pitch(fs.join(kt.script_path, 'samples/Sine440_1sec.wav')))
```

Pitch Detection

The pitch detection tries to detect the fundamental frequency of a monophonic/single note sample. It corresponds to the MIDI scale (69 = 440 Hz) and ranges from semitone 15 (~20Hz) to semitone 120 (~8.4 kHz).

detect_pitch

```
detect_pitch(file: string) -> float?
```

Analyzes `file` and returns the detected MIDI pitch or **nil** if detection fails.

Peak, RMS & Loudness detection

Loudness, Peak, and RMS functions return a value in dB, with a maximum at 0dB.

The RMS and Loudness functions are calculated over small blocks of audio. The duration of those blocks is called frame size and is expressed in seconds. The process is repeated in intervals equal to the hop size (also expressed in seconds), until it reaches the end of the sample. The functions return the overall loudest/highest value of the different blocks.

If frame size and hop size are not indicated, the default values 0.4 (frame size in seconds) and 0.1 (hop size in seconds) are applied respectively.

detect_peak

```
detect_peak(file: string) -> float?
```

Analyzes `file` and returns the detected peak level or **nil** if detection fails.

detect_rms

```
detect_rms(file: string) -> float?
detect_rms(file: string, frame_size: float, hop_size: float) -> float?
```

Analyzes `file` and returns the detected RMS level or **nil** if detection fails.

detect_loudness

```
detect_loudness(file: string) -> float?
detect_loudness(file: string, frame_size: float, hop_size: float) -> float?
```

Analyzes `file` and returns the detected loudness level or **nil** if the detection fails.

Loop Detection

```
find_loop(file: string) -> start: integer?, end: integer?
find_loop(file: string, min_start: float, max_end: float, min_length: float) ->
start: integer?, end: integer?
```

Analyzes `file` and returns the detected loop points or **nil** if detection fails.

Type Detection

Type detection is a means to determine which category a given audio sample belongs to. Currently, Kontakt Lua API supports detection of three distinct types: Sample Type, Drum Type, and Instrument Type. Sample Type is used to determine if a sample is either a drum or an instrument. Drum Type and Instrument Type both determine which drum or instrument category a sample belongs to.

sample_type

```
detect_sample_type(file: string) -> string
```

Returns the sample type of `file`. Returns one of the following types or **nil** if detection fails:

- *drum*
- *instrument*

detect_drum_type

```
detect_drum_type(file: string) -> string
```

Returns the drum type of `file`. Returns one of the following types or **nil** if detection failed.

- *kick*
- *snare*
- *hihat_closed*
- *hihat_open*
- *tom*
- *cymbal*
- *clap*
- *shaker*
- *percussion_drum*
- *percussion_other*

detect_instrument_type

```
detect_instrument_type(file: string) -> string
```

Returns the instrument type of `file`. Return one of the following types or **nil** if detection fails.

- *bass*
- *bowed_string*
- *brass*
- *flute*
- *guitar*
- *keyboard*
- *mallet*
- *organ*
- *plucked_string*
- *reed*
- *synth*
- *vocal*

13. General Information

Kontakt 7.8

New Features

- `link_resource_container()` command.

Improved Features

- When creating a resource container, typing .nkr extension is not required anymore.
- When drag and dropping a Lua script onto Kontakt's rack, it will now be listed in recently used scripts.
- If there is an error when creating a resource container, dialog will not be created any longer, instead this information will be logged to the terminal.

Fixes

- When creating a resource container, scripts from `Resources/scripts` folder would not be available to the builder for linking immediately.
- When linking a script via `set_instrument_script_linked_filename()`, the script would not be applied.