ReaLearn Reference

2024-10-22

Table of Contents

Introduction

Last update of text:	`2024-10-20 (v2.16.10)`
Last update of relevant screenshots:	`2024-10-20 (v2.16.10)`

ReaLearn is a versatile controller integration tool for REAPER. It is part of the Helgobox plug-in.

This reference provides a detailed description of ReaLearn's user interface, concepts and functionalities. Use it whenever you need an in-depth exploration of a specific feature or functionality in ReaLearn.

If you are a beginner, please use the ReaLearn Wiki instead!

This reference is targeted at users who are already familiar with the ReaLearn basics and want to look up specific information. The Wiki is perfect if you are just starting off. It guides you through the very basics and contains links to easily digestible video tutorials.

Installation

See installation instructions for Helgobox, the plug-in that contains ReaLearn.

Key concepts

This section offers brief descriptions of ReaLearn's key concepts. A solid understanding of these concepts is essential for effectively using ReaLearn, regardless of which features you plan to utilize.

Control

In ReaLearn, the term *control* typically refers to the process of triggering or adjusting something in REAPER, such as executing an action or modifying an FX parameter.

Feedback

In ReaLearn, the term *feedback* refers to controlling LEDs, motorized faders, or displays on your device in response to events in REAPER, such as a track volume change.

Controller

A *controller* is the device you use to control REAPER. It is usually a hardware device, such as a MIDI keyboard or control surface, but it can also be software, like an OSC app.

Control element

A control element is any component you can use to control something. In most cases, it's a physical part of your hardware Controller.

Examples include knobs, encoders, faders, buttons, keys, pads, pitch wheels and acceleration sensors.

Control element interaction

A control element *interaction* is the act of using a Control element.

Typically, each control element has one primary interaction type:

- *Turning* a knob
- *Pressing/releasing* a button
- *Moving* a fader

However, some control elements allow multiple interactions:

- *Moving* a touch-sensitive fader
- *Touching/releasing* a touch-sensitive fader

In this reference, Control element often implies Control element interaction, as they are usually synonymous.

Feedback element

A *feedback element* is any part of your Controller that can indicate or display information.

Examples includes LEDs, motor faders and displays.

Very frequently, control elements and feedback elements are combined:

- Button with an integrated LED
- Encoder with an LED ring
- Motorized fader

For this reason, this reference sometimes uses Control element to refer to both the Control element and the corresponding Feedback element.

Input port

To enable control, ReaLearn needs to respond to events from your Controller. It achieves this by listening to events from an *input port*, which can be a MIDI device port, an OSC port or your computer keyboard.

You can change the input port using the Input menu.

Output port

To send Feedback back to your Controller, ReaLearn transmits instructions through an *output port*, which can be a MIDI device port or an OSC port.

You can change the output port using the Output menu.

Instance

Helgobox/ReaLearn is an instrument plug-in. That means you can add multiple instances of it, just as you would add multiple instances of a synth or effect. For example, you could place one instance on the monitoring FX chain and two instances somewhere in your project.

Unit

Each ReaLearn Instance contains at least one *unit*, known as the *main unit*, but it can also contain an arbitrary number of additional units.

Units function like "mini instances" within a single ReaLearn Instance, allowing that instance to manage multiple controllers simultaneously. Each unit has its own Input port, Output port, Controller compartment, Main compartment, Controller preset, Main preset, and more.

Compartment

Each unit consists of two compartments. A compartment is a self-contained list of mappings that can be saved as an independent preset. The two compartments in each unit are:

Main compartment

This is the primary compartment. Its purpose is to define what the controller device should do, e.g., allowing a fader to control track volume or displaying the name of an FX parameter on a hardware display.

We refer to the mappings in this compartment as *main mappings* and to the presets as *main presets*.

Controller compartment

The controller compartment is optional and serves two main purposes: Describing all control elements of the controller, assigning them descriptive names and enabling Virtual control.

We refer to the mappings in this compartment as *controller mappings* and to the presets as *controller presets*.

Mapping

Each compartment contains a list of mappings.

A *mapping* connects a Control element and/or Feedback element on your Controller with an action or parameter in REAPER.

Each mapping consists of Source, Glue and Target.

Source

A *source* is the part of a Mapping that typically describes a Control element and/or Feedback element on the Controller. More generally, it can be anything that emits control values.

Examples: MIDI source, OSC source

Glue

A *glue* is the part of a Mapping that sits between Source and Target, filtering and transforming Control and Feedback streams.

Target

A *target* is the part of the Mapping that describes the thing which should be controlled and/or provides feedback data.

Examples: Track volume, cursor position, REAPER action

Learning

This section wouldn't be complete without mentioning the concept that inspired ReaLearn's name: *Learning*. Learning simply means that you press a **[Learn]** button instead of performing manual setup, saving you valuable time!

In ReaLearn, you can learn sources and targets.

Learn source

Sources can be learned by pressing the Learn source button and then touching a Control element on your controller. This saves you from the tedious job of setting up MIDI or OSC sources manually.

Learn target

Targets can be learned by pressing the Learn target button and then invoking a Target within REAPER. This saves you from choosing Target object selectors and other stuff manually.

User interface

This section provides describes the user interface elements in Helgobox relevant to ReaLearn. For details on source- or target-specific elements, refer to Sources and Targets.

Helgobox Components

Helgobox consists of two main components, each with its own user interface:

Helgobox Instrument Plug-In

This interface appears when you open the plug-in window and represents the core of ReaLearn's user interface. Most of the information in this section focuses on the instrument plug-in.

Helgobox App

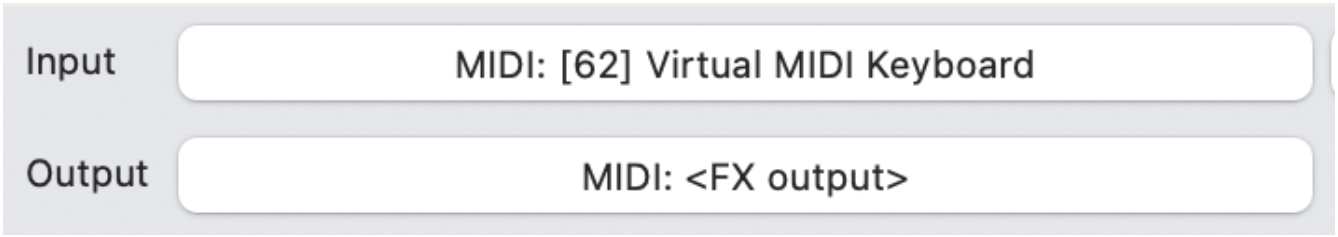
This interface is accessible via **Menu › Show app** in the plug-in. In the future, the app will also be available for mobile devices and offer remote connections to REAPER. Currently, the app only provides the interface for Playtime with one exception: ReaLearn's Projection feature.

Main panel

The main panel is what you see immediately when you open the Helgobox plug-in window.

Figure 1. Main panel

Input/output section



Use this section to connect ReaLearn to a specific controller.

Also see Best practices for setting input and output.

Input menu

Allows you to select the Input port to which this ReaLearn unit listens. ReaLearn works with MIDI or OSC input. In addition, it can listen to keyboard input.

MIDI: <FX input>

When selected, ReaLearn captures all MIDI events that flow into this ReaLearn VSTi FX instance (= track MIDI path). This is the default selection.

MIDI: *Some input device*

When selected, ReaLearn captures events from the given MIDI device directly, before it reaches REAPER's tracks.

This will only work if *Enable input from this device* is checked for the selected MIDI input device in REAPER's preferences (**Options › Settings › MIDI Inputs**). The device name is prefixed with the device ID, which also shows up in the REAPER preferences.

Unavailable MIDI input devices

This submenu contains MIDI input devices which are currently disconnected or not enabled in the REAPER preferences.

OSC: *Some device*

When selected, ReaLearn captures events from the given OSC device. Before any device shows up here, you need to add it via Manage OSC devices.

Unavailable OSC devices

This submenu contains OSC devices for which control is currently disabled.

Manage OSC devices

Allows one to display and modify the list of OSC devices (globally).

<New>

Opens a dialog window for adding a new OSC device. See OSC device dialog.

Some OSC device

Edit

Lets you edit an existing device. See OSC device dialog.

Remove

Removes the device. This is a global action. If you remove a device, all existing ReaLearn instances which use this device will point to a device that doesn't exist anymore.

Enabled for control

If you disable this, ReaLearn will stop listening to this device. This can save resources, so you should do this with each device that is not in use (as an alternative for removing it forever).

Enabled for feedback

If you disable this, ReaLearn won't send anything to this device.

Can deal with bundles

By default, ReaLearn aggregates multiple OSC messages into so-called OSC bundles. Some devices (e.g. from Behringer) can't deal with OSC bundles. Untick the checkbox in this case

and ReaLearn will send single OSC messages.

Computer keyboard

This is a checkbox. If enabled, this ReaLearn instance will additionally listen to key press and release events.

Output menu

Here you can choose to which Output port ReaLearn should send MIDI/OSC Feedback.

<None>

This means, no feedback will be sent at all. This is the default.

MIDI: <FX output>

This makes feedback MIDI events stream down to the next FX in the chain or to the track's hardware MIDI output.

MIDI: *Some output device*

If selected, ReaLearn will send feedback to the given MIDI output device. This only works if *Enable output to this device* is checked in REAPER's preferences (**Options › Settings › MIDI Outputs**).

OSC: *Some device*

When selected, ReaLearn will send feedback to the given OSC device. Before any device shows up here, you need to add it via Manage OSC devices.

Unavailable OSC devices

This submenu contains OSC devices for which feedback is currently disabled.

Manage OSC devices

See Manage OSC devices in the input section of the menu.

OSC device dialog

The OSC device dialog lets you edit the settings of a ReaLearn OSC device and can be opened via Manage OSC devices.

Name

A descriptive name of the device, e.g. "TouchOSC on my Android phone".

Local port

Required for control. The UDP port on which ReaLearn should listen for OSC control messages.

This port must be reserved exclusively for ReaLearn! If you already use this port in another application (e.g. in REAPER's own OSC control surface) it won't work and ReaLearn will bless you with an "unable to connect" message in the "Input" dropdown.

Device host

Required for feedback only. It's the IP address of the OSC device to which ReaLearn should send

feedback messages. This address is usually displayed on your OSC device (e.g. as "Local IP address"). When targeting an OSC software that runs on the same computer as REAPER and ReaLearn, enter the special IP address `127.0.0.1` (or `localhost`).

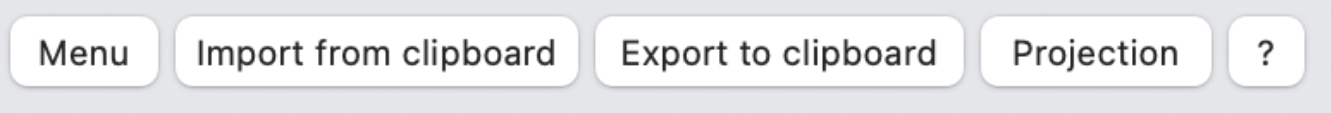
When you configure your OSC connection on the device side, you must provide a *host* as well. There you should enter the IP address of the computer which runs REAPER and ReaLearn. You can easily find it by pressing the Projection button in ReaLearn, choosing the old browser-based projection and scrolling down a bit. It's the value next to **Host** and should start with `192.168.`.

Device port

Required for feedback only. The UDP port on which the OSC device listens for OSC feedback messages.

All OSC device configurations will be saved in the REAPER resource directory (**Options › Show REAPER resource path in explorer/finder**) in the file `Helgoboss/ReaLearn/osc.json`.

Menu bar



Menu button

This opens the main menu of Helgobox/ReaLearn. The same menu opens when you right-click an empty area.

It provides the following entries.

Copy listed mappings

Copies all mappings that are visible in the current mapping list (respecting group, search field and filters) to the clipboard. You can insert them by opening the right-click menu of a row panel.

Paste mappings (replace all in group)

Replaces all mappings in the current group with the mappings in the clipboard.

Modify multiple mappings

Auto-name listed mappings

Clears the names of all listed mappings. As a result, ReaLearn will display dynamically auto-generated mapping names instead.

Name listed mappings after source

Sets the names of each listed mapping to the first line of its source label.

Make sources of all main mappings virtual

Attempts to make the sources in the main compartment virtual by matching them with the sources in the controller compartment. This is useful if you already learned a bunch of MIDI/OSC/keyboard sources in the main compartment, just to find out later that you would like to use a controller preset that exposes virtual sources.

Make targets of listed mappings sticky

Changes the targets of all currently listed mappings so that they use *sticky* object selectors by attempting to resolve the objects from non-sticky selectors. See Sticky selectors.

Make targets of listed mappings non-sticky

Changes the targets of all currently listed mappings so that they use the desired non-sticky object selectors. You can see this in action in tutorial video 9.

The general procedure is:

1. Choose the desired track selector
2. Choose the desired FX selector
3. ReaLearn will change the selectors for all targets where this is applicable

Move listed mappings to group

Moves all currently listed mappings to the specified group. Useful in combination with text search.

Advanced

Provides access to expert features.

Copy listed mappings as Lua

Like Copy listed mappings but generates Lua/Luau code instead.

Copy listed mappings as Lua (include default values)

Generates Lua/Luau code that contains even those properties that correspond to ReaLearn's defaults.

Paste from Lua (replace all in group)

Like Paste mappings (replace all in group) but treats the clipboard content as Luau code.

Dry-run Lua script from clipboard

Executes the Luau code in the clipboard and opens the returned data structure in a text editor. See Import from clipboard button to learn in which situations this can be useful.

Freeze Playtime matrix

Don't use this, this feature is not ready yet!

Compartment parameters

This shows all parameters of the current compartment and makes it possible to customize them. See Compartment parameter to learn what such parameters are good for.

Each parameter provides the following customization options:

Name

Changes the name of this parameter.

Value count

Lets you enter the maximum number of values. This automatically turn this parameter into a discrete parameter. See Continuous vs. discrete compartment parameters.

Compartment presets

Create compartment preset workspace

Exports the Luau SDK files of this ReaLearn version into a new compartment preset workspace directory with a random name. See section Writing presets with Luau for details.

Create compartment preset workspace (including factory presets)

Exports the Luau SDK files of this ReaLearn version and all factory presets for this compartment into a new compartment preset workspace directory with a random name. See section Writing presets with Luau for details.

Open compartment preset folder

Opens the ReaLearn preset folder for this compartment in a file manager.

Reload all compartment presets from disk

If you made direct changes to preset files, you should press this to reflect these changes in the compartment preset lists of all open ReaLearn instances (reloads all preset files in this compartment).

This **will not** apply an adjusted preset to the current compartment, it will just reload the list. If you want to apply a preset that has been changed on disk, you need to reload it by selecting it in the preset dropdown once again!

Edit compartment-wide Lua code

Allows you to edit the compartment-wide Lua code. See section Compartment-wide Lua code.

Unit options

Auto-correct settings

By default, whenever you change something in ReaLearn, it tries to figure out if your combination of settings makes sense. If not, it makes an adjustment. This autocorrection is usually helpful. You can disable this checkbox if you don't like this behavior.

Send feedback only if track armed

Here you can tell ReaLearn to only send feedback when the track is armed.

At the moment, this can only be unchecked if ReaLearn is on the normal FX chain. If it's on the input FX chain, unarming forcefully disables feedback because REAPER generally excludes input FX from audio/MIDI processing while a track is unarmed (this is subject to change in the future).

Normally, you don't have to touch this because Auto-correct settings automatically chooses a reasonable default, depending on which input is selected:

- If input is set to MIDI: <FX input>, it enables this option so that ReaLearn only sends feedback if the track is armed. Rationale: Unarming will naturally disable control, so

disabling feedback is just consequent.

- If input is set to a specific MIDI or OSC device, it disables this option in order to allow feedback even when unarmed.

Reset feedback when releasing source

When using ReaLearn the normal way, it's usually desired that feedback is reset when the corresponding sources are not in use anymore (e.g. lights are switched off, displays are cleared, motor faders are pulled down).

You can prevent this unit from doing that by disabling this option. This can be useful e.g. when using REAPER/ReaLearn just in feedback direction, in order to take control of a hardware device (= using ReaLearn the other way around, "controlling from target to source").

Make unit superior

Makes this unit superior. See Superior units to learn more about this feature.

Use unit-wide FX-to-preset links only

By default, unit-wide links are applied *in addition* to the global links and take precedence over the global ones. This checkbox makes sure that only unit-wide links are used.

Stay active when project in background

Determines if and under which conditions this ReaLearn unit should stay active when the containing project tab is not the active one. Applies to in-project ReaLearn instances only, not to monitoring FX instances!

Never

Will only be active when its project tab is active.

Only if background project is running

Follows REAPER's project tab settings ("Run background projects" and "Run stopped background projects").

Always (more or less)

Attempts to stay active no matter what. Please note that this is technically not always possible when input is set to MIDI: <FX input> or output to MIDI: <FX output>, when the background project is not running.

Unit-wide FX-to-preset links

Manage a unit-wide list of links from FX (plug-ins or JS effects) to ReaLearn main compartment presets. Covered in video tutorial 10.

Add link from last focused FX to preset

This lets you link whatever FX window was focused before focusing ReaLearn, to an arbitrary main compartment preset. This only works if an FX has been focused before.

Arbitrary FX ID

If you have added a link already, you will see it here in the list. What you see, is the so-called

FX ID, which by default simply corresponds to the plug-in's original name (e.g. **Name: VSTi: ReaSynth (Cockos) | File: - | Preset: -**).

<Edit FX ID…>

With this, you can edit the FX ID manually. See FX ID dialog for details.

<Remove link>

(Globally) removes this FX-to-preset link.

Arbitrary main preset

The rest of the submenu tells you to which main preset the FX ID is linked. You can change the linked preset by clicking another one.

Logging

Log debug info (now)

Logs some information about ReaLearn's internal state. Can be interesting for investigating bugs or understanding how this plug-in works.

Log real control messages

When enabled, all incoming MIDI messages, OSC messages or key pressed will be logged to the console. See Logging of real control messages.

Log virtual control messages

When enabled, all triggered virtual control elements and their values will be logged (see Controller compartment).

Log target control

When enabled, all target invocations (parameter changes etc.) will be logged.

Log virtual feedback messages

When enabled, all feedback events to virtual control elements will be logged (see Controller compartment).

Log real feedback messages

When enabled, all outgoing MIDI or OSC messages will be logged to the console. See Logging of real feedback messages.

Send feedback now

Usually ReaLearn sends feedback whenever something changed to keep the LEDs or motorized faders of your controller in sync with REAPER at all times. There might be situations where it doesn't work though. In this case you can send feedback manually using this button.

Instance options

Enable global control

If you enable this option, this Helgobox instance will start to automatically add/remove units based on connected controllers. See Auto units.

Open Pot Browser

This will open Pot Browser. See Pot Browser for details.

Show App

Shows the Helgobox App associated with this Helgobox instance.

Close App

Closes the Helgobox App associated with this Helgobox instance.

User interface

Background colors

Enables/disables the usage of background colors in the ReaLearn user interface (enabled by default).

Server

ReaLearn features a built-in server which allows the Companion App (and in future also the Helgobox App) to connect to ReaLearn. The server runs globally, not per instance!

Enable and start!

This starts the server and makes sure it will automatically be started next time you use ReaLearn.

Disable and stop!

This stops the server and makes sure it will not be started next time you use ReaLearn.

Add firewall rule

Attempts to add a firewall rule for making the server accessible from other devices or displays instructions how to do it.

Global FX-to-preset links

Allows you to manage global FX-to-preset links. Works exactly as the Unit-wide FX-to-preset links menu.

FX ID dialog

The FX ID dialog is used to edit which properties of a FX trigger a preset change. It is opened via menu action <Edit FX ID...>.

FX name

Allows you to adjust the (original) plug-in name that triggers the preset change.

FX file name

Allows you to adjust the plug-in file name that triggers the preset change.

FX preset name

Maybe the FX name or file name is not enough for you to decide which preset you want to load. You can add a preset name as additional criteria.

Example 1. Samplers

> If you have a sampler, you can load different ReaLearn presets depending on which sample library is loaded into your sampler. Just add two links with the same FX file name (e.g. `Kontakt 5.dll`) but different preset names.

All above-mentioned fields support wildcards. You can use `*` for matching zero or arbitrary many characters and `?` for matching exactly one arbitrary character.

Example 2. Matching both VST2 and VST3 plug-ins

> Instead of relying on the original plug-in name you could match plug-ins with similar file names (e.g. VST2 and VST3 at once): `Pianoteq 7 STAGE.*` would match both `Pianoteq 7 STAGE.dll` (VST2) and `Pianoteq 7 STAGE.vst3` (VST3).

Export to clipboard button

Pressing the export button allows you to copy ReaLearn's settings to the clipboard so you can import them in another instance/unit or edit them in a text editor. See Import/export.

Export instance as JSON

Copies a *complete* dump of this Instance to the clipboard in JSON format.

Export main/controller compartment as JSON

Copies a dump of the currently visible compartment to the clipboard. It contains about the same data that a compartment preset would contain.

Export main/controller compartment as Lua

Copies a dump of the currently visible compartment to the clipboard as Lua/Luau code (ReaLearn Script). This form of Lua/Luau export skips properties that correspond to ReaLearn's default values, resulting in a minimal result. Perfect for pasting into a forum or programming ReaLearn with focus on only those properties that matter to you.

Export main/controller compartment as Lua (include default values)

This Lua/Luau export includes even those properties that correspond to ReaLearn's default values, resulting in more text. This gives you the perfect starting point if you want to extensively modify the current compartment (using the Luau programming language) or build a compartment from scratch, using even properties that you haven't touched yet in the user interface!

Import from clipboard button

Pressing the import button does the opposite: It restores whatever ReaLearn dump is currently in the clipboard. It supports JSON or Luau. See Import/export.

Projection button

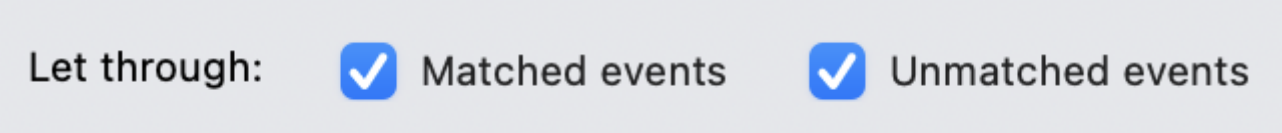
Click this button to enter ReaLearn's Projection feature. You can choose between the old browser-

based projection (which is going to disappear soon) and the new projection that is part of the Helgobox App (but not yet fully functional). Hopefully, the transition to the Helgobox App, including mobile versions of that App, will soon be finished.

Help button (?)

Provides links to the reference and other documentation.

Let through section



See Letting through events.

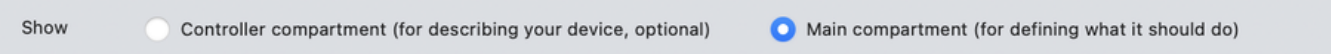
Matched

If checked, even events matched by at least one mapping are let through. If unchecked (default), such events are filtered out.

Unmatched

If checked (default), mappings that are not matched by any mappings are let through. If unchecked, such events are filtered out.

Show section



This lets you choose which mapping compartment is displayed. See Compartment.

Preset section

Preset menu

This menu makes it possible to load compartment presets for the currently shown compartment. If you select a preset in this list, its corresponding mappings will be loaded and immediately get active.

The following menu entries are available:

<None>

This entry is selected by default. It means that no particular preset is active.

Selecting this will always clear the complete compartment, including all mappings!

Factory

Contains available factory presets.

User (…)

Contains available user presets. Multiple of such submenus may exist. Each one represents a different preset namespace/workspace. The namespace named after you (macOS/Linux/Windows username) is your personal user namespace.

For more information about preset workspaces/namespaces, see Writing presets with Luau.

User (Unsorted)

This submenu contains top-level presets which are not part of a particular preset namespace/workspace. This was common in older versions of ReaLearn, when namespaces/workspaces were not yet available.

Save button

If you made changes to a user preset, you can save them by pressing this button.

Save as… button

This allows you to save all currently visible mappings as a new preset. Please choose a descriptive name.

Delete button

This permanently deletes the currently chosen user preset.

Auto-load button

Activates or deactivates Auto-load mode for this ReaLearn unit. This button is only available for the Main compartment because auto-load is only about loading main presets.

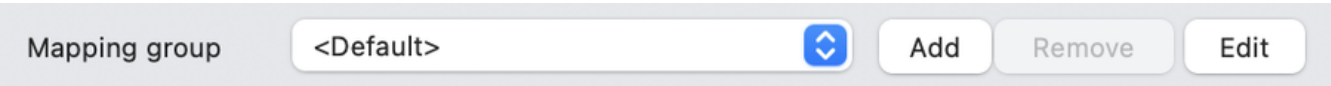
Off

Disables auto-load mode (the default).

Based on unit FX

Switches auto-load mode on, letting ReaLearn decide about which main preset to load depending on the currently active Unit FX.

Mapping group section



Group menu

See Mapping group.

The group menu contains the following options:

<All>

Displays all mappings in the compartment, no matter to which group they belong. In this view,

you will see the name of the group on the right side of a mapping row.

<Default>

Displays mappings that belong to the *default* group. This is where mappings end up if you don't care about grouping. This is a special group that can't be removed.

Custom group

Displays all mappings in your custom group.

Add button

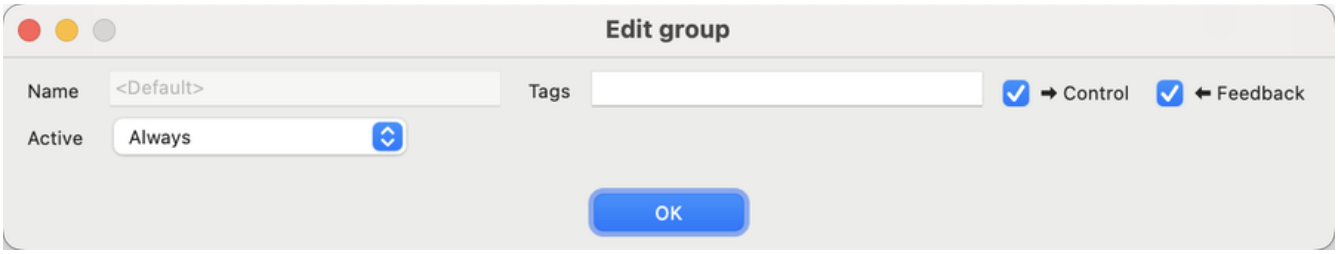
Allows you to add a group and give it a specific name.

Remove button

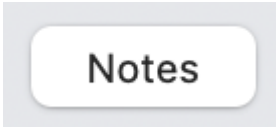
Removes the currently displayed group. It will ask you if you want to remove all the mappings in that group as well. Alternatively they will automatically be moved to the default group.

Edit button

Opens the group panel, which allows you to change Group properties.

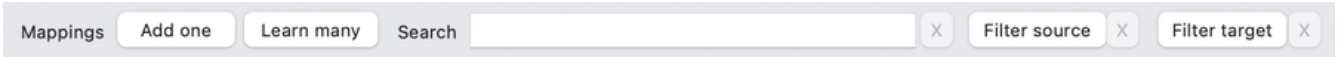


Notes button



Allows you to save custom notes/comments for the current compartment. These notes are also included in compartment presets.

Mappings toolbar



Add one button

Adds a new mapping at the end of the current mapping list.

Learn many button

Allows you to add and learn many new mappings in a convenient batch mode. Click this button and follow the on-screen instructions. Click *Stop* when you are finished with your bulk learning strike.

Search field

Enter text here in order to display just mappings whose name matches the text.

You can search for mappings that have a certain tag by entering the tag name prefixed with the hash character `#`. For example, you can search for all mappings tagged with the tag `mixing` by entering `#mixing`.

The search expression also supports wildcards `*` and `?` for doing blurry searches. `*` stands for zero or more arbitrary characters and `?` stands for one arbitrary character.

Filter source button

When you press this button, ReaLearn will start listening to incoming MIDI/OSC events and temporarily disable all target control. You can play around freely on your controller without having to worry about messing up target parameters. Whenever ReaLearn detects a valid source, it will filter the mapping list by showing only mappings which have that source.

This is a great way to find out what a specific knob/fader/button etc. is mapped to. Please note that the list can end up empty (if no mapping has that source).

As soon as you press **[Stop]**, the current filter setting will get locked. This in turn is useful for temporarily focusing on mappings with a particular source.

When you are done, and you want to see all mappings again, press the **[X]** button to the right, which clears the filter.

Before you freak out thinking that ReaLearn doesn't work anymore because it won't let you control targets, have a quick look at this button. ReaLearn might still be in "filter source" mode. Then just calm down and press **[Stop]**. It's easy to forget.

Filter target button

If you want to find out what mappings exist for a particular target, press this button and touch something in REAPER.

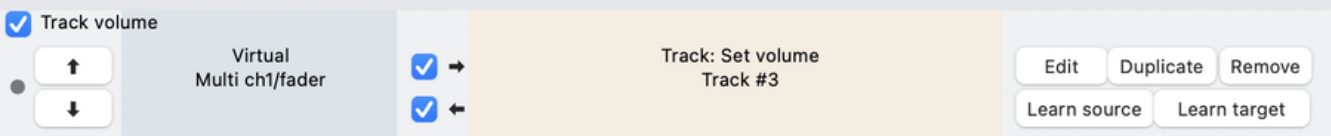
As soon as you have touched a valid target, the list will show all mappings with that target.

Unlike Filter source button, ReaLearn will automatically stop learning as soon as a target was touched.

Press the **[X]** button to clear the filter and show all mappings again.

The mapping rows area consists of multiple mapping rows. One for each mapping.

Mapping row



Each mapping row represents one ReaLearn mapping.

The mapping, source and target labels of a mapping row are greyed out whenever the mapping is *off*. See Mapping.

Mapping-enabled checkbox

This checkbox at the top left of the mapping row enables or disables the mapping as a whole.

Activity indicator (●)

This indicator at the very left of the mapping row lights up on incoming control messages whenever they match the mapping source.

Attention: This doesn't necessarily mean that the message will reach the target, although it often does. There are certain settings in the Glue section section which allow you to filter messages even they matched the source (e.g. Source Min/Max controls).

Up/down buttons

Use these buttons to move this mapping up or down the list.

Control/feedback-enabled checkboxes (→/←)

Use these checkboxes to enable/disable control and/or feedback for this mapping. Disabling both has the same effect as disabling the mapping as a whole.

Edit button

Opens the mapping panel for this mapping.

Duplicate button

Creates a new mapping just like this one right below.

Remove button

Removes this mapping from the list.

Learn source button

Starts or stops learning the source of this mapping. See Learn source.

Learn target button

Starts or stops learning the target of this mapping.

Learning a target that is currently being automated is not possible at the moment because ReaLearn can't know if the value change notification is coming from the automation or your touch interaction.

Right-click menu

Each mapping row provides a right-click menu for accessing the following functions:

Copy
: Copies this mapping to the clipboard.

Paste (replace)
: Replaces this mapping with the mapping in the clipboard. If the clipboard contains just a part of a mapping (source, mode or target), then just this part gets replaced.

Paste (insert below)
: Creates a new mapping that's like the mapping in the clipboard and places it below this mapping.

Copy part
: Copies just a part of the mapping (activation condition, source, mode or target).

Move to group
: Lets you move this mapping to another mapping group.

Advanced
: Provides access to expert features.

  Copy as Lua
  : Copies this mapping as Lua/Luau code. This is an indispensable tool if you want to build your mappings in Luau because it gives you a readily executable code snippet that you can adjust as desired.

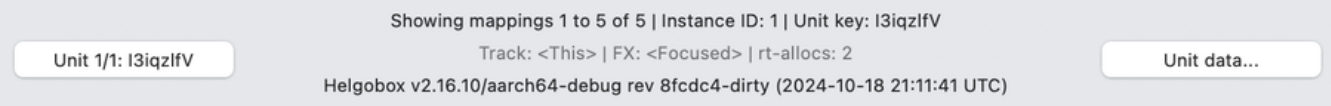
  Copy as Lua (include default values)
  : Includes even default values.

  Paste from Lua (replaces)
  : Like *Paste (replace)* but treats the clipboard content as Luau code.

  Paste from Lua (insert below)
  : Like *Paste (insert below)* but treats the clipboard content as Luau code.

  Log debug info (now)
  : Logs debug information about this particular mapping.

Bottom section



Unit menu

Press the button will reveal a menu with the following actions to manage Units:

Remove current unit

Removes the current unit. This can't be undone!

List of units

Switch to an arbitrary unit by clicking on it.

Add unit

Adds a new unit within this instance. The new unit will automatically be named after the randomly-generated unit key. You can change the name by pressing the Unit data… button.

Info area

In the center you can see the info area, made up of 3 rows:

Row 1

- The current scroll position.
- The Instance ID of this Helgobox Instance.
- The Unit key of the currently visible ReaLearn Unit.
- Unit tag assigned to this ReaLearn Unit.

Row 2

- Information about the current Unit track and Unit FX.
- Information whether control and/or feedback is currently inactive unit-wide.

Row 3

- Information about what version of Helgobox is running.

Unit data… button

Press this button to change various key-value data of this ReaLearn unit as a whole.

Unit key…

Allows you to change the Unit key of this Unit.

Unit name

The display name of this Unit.

Tags

Lets you assign Unit tag to this Unit as a comma-separated list.

Mapping panel

When you press the **[Edit]** button of a mapping row, the *mapping panel* appears, which lets you look at the corresponding mapping in detail and modify it.

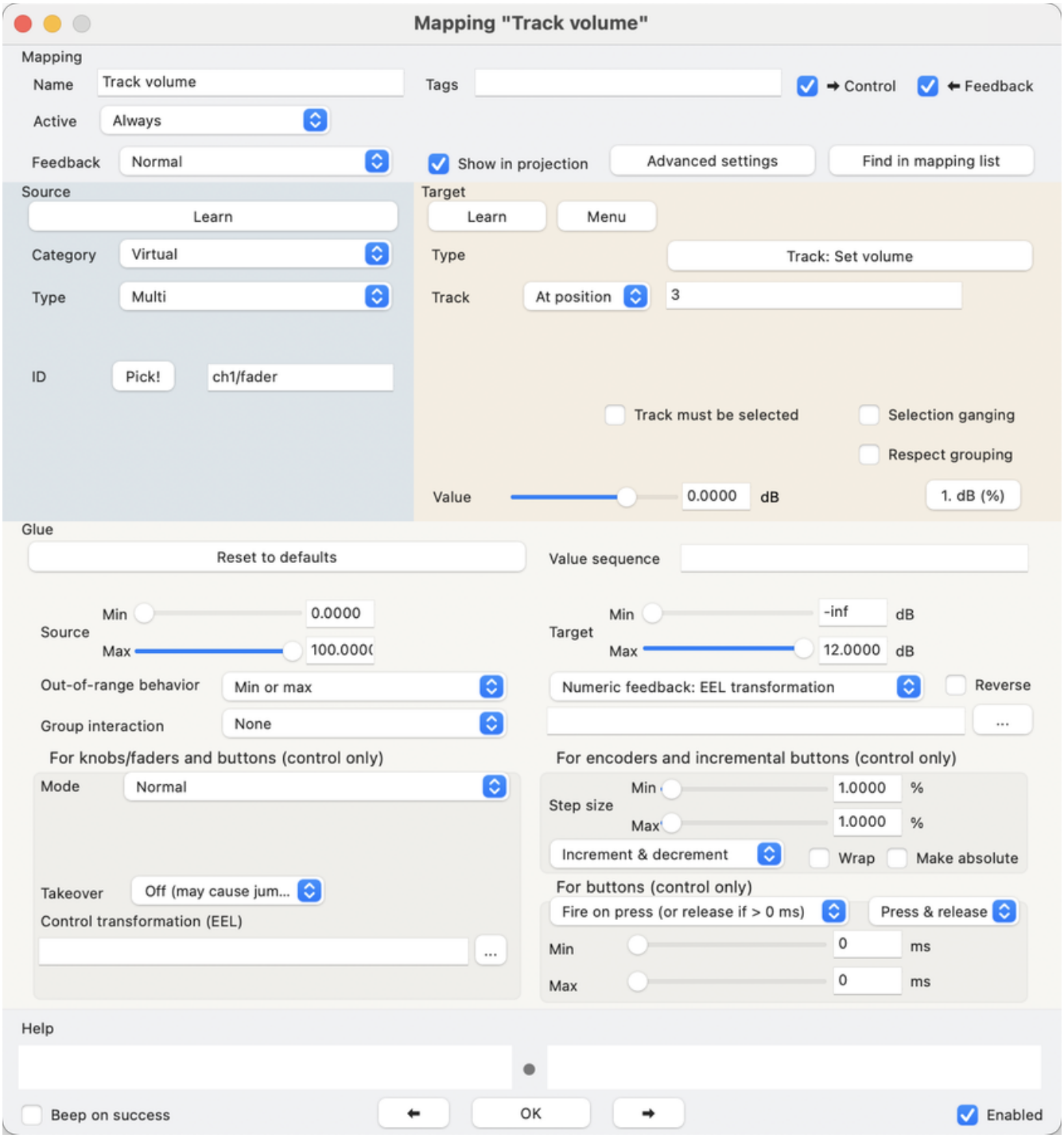
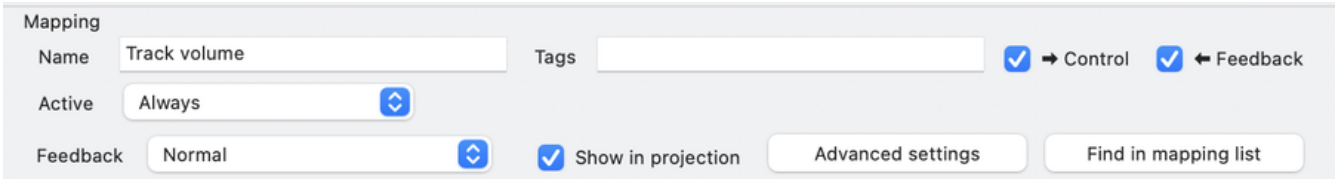


Figure 2. Mapping panel

Top section



This section provides the following mapping-related elements.

Name field

Here you can enter a descriptive name for the mapping. This is especially useful in combination with the search function if there are many mappings to keep track of.

If you clear the name, ReaLearn will name the mapping automatically based on its target.

Tags field

Use this to assign arbitrary mapping tags to this mapping (comma-separated).

Control-enabled checkbox (→)

Use this to enable/disable control for this mapping.

Feedback-enabled checkbox (←)

Use this to enable/disable feedback for this mapping.

Disabling both control and feedback has the same effect as disabling the mapping as a whole.

Active menu

This dropdown can be used to enable Conditional activation for this mapping.

Feedback menu

Normal

Makes ReaLearn send feedback whenever the target value changes. This is the recommended option in most cases.

Prevent echo feedback

This option mainly exists for motorized faders that don't like getting feedback while being moved. If checked, ReaLearn won't send feedback if the target value change was caused by incoming source events of this mapping. However, it will still send feedback if the target value change was caused by something else, e.g. a mouse action within REAPER itself.

Send feedback after control

This checkbox mainly exists for "fixing" controllers which allow their LEDs to be controlled via incoming MIDI/OSC *but at the same time* insist on controlling these LEDs themselves. For example, some Behringer X-Touch Compact buttons exhibit this behavior in MIDI mode. Such a behavior can lead to wrong LED states which don't reflect the actual state in REAPER.

If this option is not selected (the normal case and recommended for most controllers), ReaLearn will send feedback to the controller *only* if the target value has changed. For example, if you use a button to toggle a target value on and off, the target value will change only when pressing the button, not when releasing it. As a consequence, feedback will be sent only when pressing the button, not when releasing it.

If this option is selected, ReaLearn will send feedback even after releasing the button - although the target value has not been changed by it.

Another case where this option comes in handy is if you use a target which doesn't support proper feedback because REAPER doesn't notify ReaLearn about value changes (e.g. "Track FX all enable"), and you have "Poll for feedback" disabled. By choosing this option, ReaLearn will send feedback whenever the target value change was caused by ReaLearn itself, which improves the situation at least a bit.

Show in projection checkbox

When unticked, this mapping will not show up in Projection. Useful e.g. for feedback-only mappings or buttons with multiple assignments.

Advanced settings button

This button is for experts. There are some advanced mapping-related settings in ReaLearn that are not adjustable via its graphical user interface but only by writing text-based configuration. Pressing this button should open a small window in which you can write the configuration for this mapping.

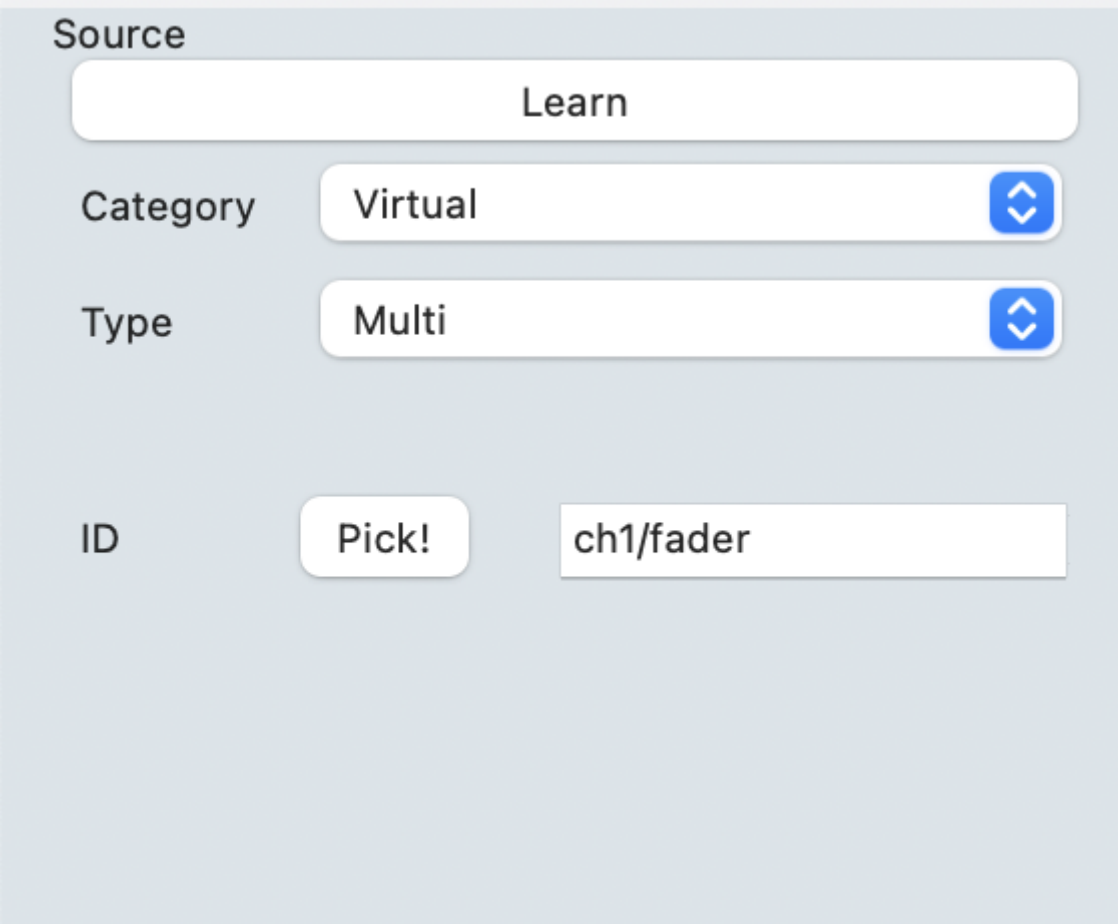
If the button label ends with a number, that number denotes the number of top-level configuration properties set for that mapping. That way you can immediately see if a mapping has advanced settings or not.

You can learn more about the available properties in the section Advanced settings dialog.

Find in mapping list button

Scrolls the mapping rows panel so that the corresponding mapping row for this mapping gets visible.

Source section



All Sources share the following UI elements.

Learn button

Starts or stops learning the Source of this mapping.

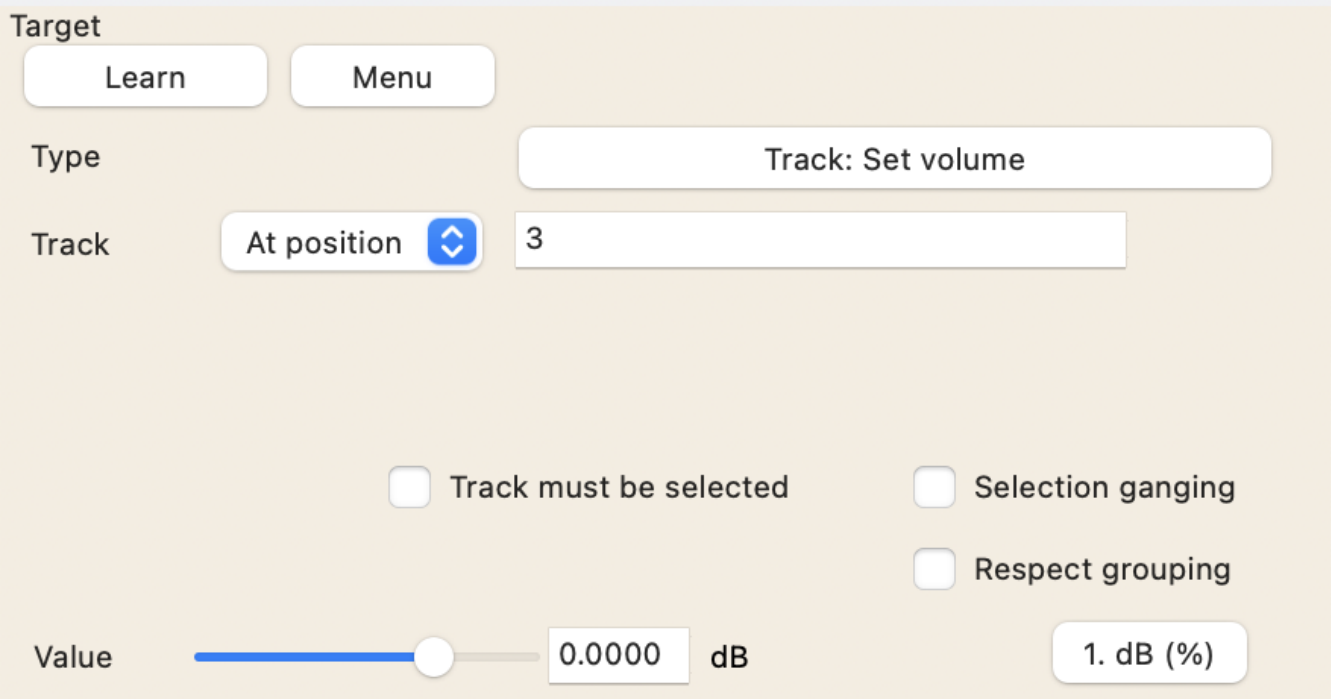
Category menu

Lets you choose the source category.

Type menu

Lets you choose the source type within that category. See Sources.

Target section



All Targets share the following UI elements.

Learn button

Starts or stops learning the target of this mapping.

Menu

Opens a small menu related to the target section:

Pick recently touched target (by type)

Gives you a list of recently touched parameters or executed actions in REAPER. When you click one of it, the target will be populated accordingly. It's an alternative to **[Learn]**.

Please note that not all targets can be picked that way, some have to be configured manually.

Go there (if supported)

If applicable, this makes the target of this mapping visible in REAPER. E.g. if the target is a track FX parameter, the corresponding track FX window will be displayed.

Category menu

Lets you choose the target category.

Type menu

Lets you choose a target type within that category. See Targets.

Value section and field

Reflects the current value of this mapping target and lets you change it (either via slider and text field or via buttons, depending on the target character).

If the target can't be resolved at the moment, it will show "Target currently inactive!" instead.

Display unit button

On the right side of the current value you will see a button with a label such as `1. dB (%)`. This button displays the currently selected target unit (unrelated to the Unit concept) which is used for displaying and entering target values.

The number in the parentheses denotes the unit which is used for displaying and entering target step sizes.

Clicking the button switches between available target units. Currently, there are two options:

(1) Use native target units

Uses the target-specific unit, e.g. dB for volume targets. If the target doesn't have any specific units, it will be displayed as `1. - (-)`.

(2) Use percentages

Uses percentages for everything, which can be nice to get a uniform way of displaying/entering values instead of having to deal with the sometimes clunky target-specific units.

Common elements for track targets

When choosing a track, the following additional elements are available.

Track must be selected checkbox

If checked, this mapping will be active only if the track set in *Track* is currently selected. See Target activation condition.

Selection ganging checkbox

If checked and if the track in question is selected, all other selected tracks will be adjusted as well. This uses REAPER's built-in selection-ganging feature and therefore should behave exactly like it.

Respect grouping checkbox

If checked, track grouping will be taken into account when adjusting the value. This uses REAPER's built-in track grouping feature and therefore should behave exactly like it.

In older REAPER versions (< 6.69+dev1102), this can only be enabled together with selection ganging when using it on volume, pan or width targets.

Common elements for on/off targets

Targets which control an on/off-style property of tracks (e.g. Target "Track: Solo/unsolo") additionally provide the following elements.

Exclusive menu

By default, this is set to No.

No

Makes the track target affect just this track.

Within project

Switches the property on (off) for this track and off (on) for all other tracks in the project.

Within folder

Switches the property on (off) for this track and off (on) for all other tracks in the same folder and same level.

Within project (on only)

Variation of *Within project* that applies exclusivity only when switching the property on for this track. In other words, it never switches the property on for other tracks.

Within folder (on only)

Variation of *Within folder* that applies exclusivity only when switching the property on for this track. In other words, it never switches the property on for other tracks.

Common elements for send targets

Only available for targets that work on a send/receive.

Kind menu

The kind of send/receive that you want to control.

Send

Send from the track above to another track of your choice. If you choose Particular selector, ReaLearn will memorize the ID of the destination track. That way you will still control the correct send even if you delete another send in that track.

Receive

Receive from another track of your choice to the track above (opposite direction of send). If you choose the Particular selector selector, ReaLearn will memorize the ID of the source track.

Output

Send from the track above to a hardware output. Please note that with hardware outputs, Particular selector is the same as At position selector because hardware outputs don't have unique IDs.

Send/Receive/Output section

This lets you choose the actual send/receive/output.

Common elements for FX targets

The following elements and selectors are available for targets associated with a particular FX instance.

FX section

The FX instance associated with this target. ReaLearn will search for the FX in the output or input FX chain of the above selected track.

Input FX checkbox

If unchecked, the *FX* dropdown will show FX instances in the track's normal FX chain. If checked, it will show FX instances in the track's input FX chain.

Monitoring FX checkbox

This appears instead of the input FX checkbox if you select track `<Master>`. If you check this, you can target FX instances on REAPER's global monitoring FX chain.

Because of a limitation in the REAPER API, learning and feedback for monitoring FX doesn't work!

FX must have focus checkbox

If checked, this mapping will be active only if the selected FX instance is currently *focused*.

If the FX instance is displayed in a floating window, *focused* means that the floating window is active. If it's displayed within the FX chain window, *focused* means that the FX chain window is currently open and the FX instance is the currently selected FX in that FX chain.

Of course, this flag doesn't have any effect if you chose the Focused selector.

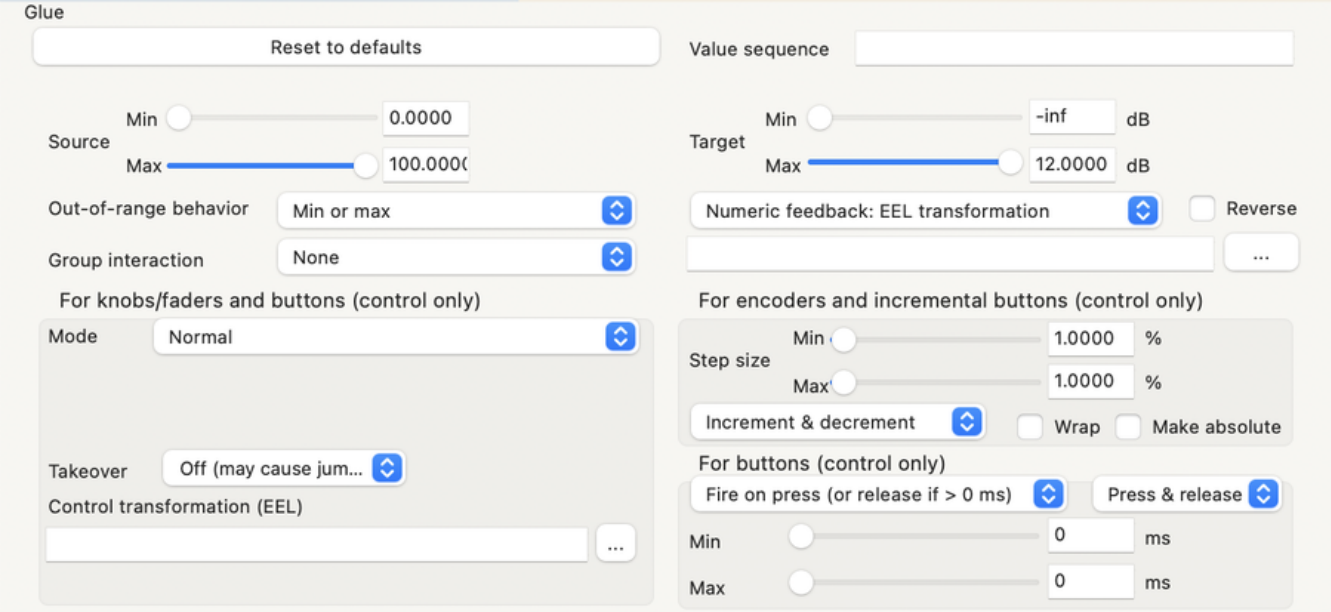
Common elements for pollable targets

The following elements are available only for the few targets that might need polling (= regular value querying) in order to support automatic feedback in all cases.

Poll for feedback checkbox

Enables or disables Target value polling. In the probably rare case that the polling causes performance issues, you can untick this checkbox.

Glue section



The Glue section is divided into several subsections some of which make sense for all kinds of sources and others only for some. Having so many settings available at the same time can be a bit daunting. ReaLearn helps you by hiding settings that don't make sense in the current context.

It shows or hides them based on criteria like this:

- Is control and/or feedback enabled for the mapping?
- What are the characteristics of the source and target?
- What's the current setting of Mode ("Absolute mode") menu and Make absolute?

Reset to defaults button

Resets the settings to some sensible defaults.

Reverse checkbox

Control →
: inverses incoming absolute control values and changes the direction of incoming relative control values

Feedback ←
: inverses the feedback value

Depending on the context, this can have different effects:

Element	Direction	Effect
Momentary button	Control →	Switches the target off when pressed and on when released
Range element (fader, knob, …)	Control →	The higher the fader position, the lower the target value

Element	Direction	Effect
Incremental button	Control →	Decreases the target value on press instead of increasing it
Rotary endless encoder	Control →	Decreases the target value when turning clockwise and decreases it when turning counter-clockwise
LED	Feedback ←	Uses **off** LED color if target is on and **on** LED color if target is off
Value indicator (LED ring, motor fader, …)	Feedback ←	The higher the target value, the lower the indicated value

Target Min/Max controls

Control →

sets the controlled value range of the target, making sure that the target value ends up within that range

Feedback ←

sets the observed range of feedback values coming from the target

Example 3. Squeezing the track volume value range

If you set this to "-6 dB to 0 dB" for a *Track volume* target, the volume will always stay within that dB range when controlled via this mapping. However, it wouldn't prevent the volume from exceeding that range if changed e.g. in REAPER itself.

This setting can be used with all targets that work with absolute control values (all targets except Target "Project: Invoke REAPER action" with relative Invocation type).

Value sequence field

Control →

sets discrete target values to step through

Allows you to define a Target value sequence. All values are entered comma-separated using the target unit specified with the Display unit button.

You can provide only one of Target Min/Max controls or Value sequence field.

Group interaction

Control →

allows controlling not only this mapping but also all other mappings in the same group

See Mapping group.

If you want to control *other* mappings only and not *this* mapping, just pick a target that doesn't have any effect, for example the Target "ReaLearn: Dummy".

None

Switches group interaction off. This is the default. Incoming control events will just affect *this* mapping, not others.

Same control

This will broadcast any incoming control value to all other mappings in the same group. The glue section of this mapping will be ignored when controlling the other mappings. The glue sections of the other mappings will be respected, including the Source Min/Max controls.

Same target value

This will set the target value of each other mapping in the same group to the target value of this mapping. Nice: It will respect the Target Min/Max controls of both this mapping and the other mappings. All other settings of the glue section will not be processed. Needless to say, this kind of control is always absolute, which means it can lead to parameter jumps. Therefore, it's most suited for on/off targets. If you don't like this, choose Same control instead.

Inverse control

This is like Same control but broadcasts the *inverse* of the incoming control value.

Inverse target value

This is like Same target value but sets the target values of the other mappings to the *inverse* value. This is very useful in practice with buttons because it essentially gives you exclusivity within one group. It's a great alternative to the Exclusive menu which is available for some targets. Unlike the latter, Inverse target value allows for exclusivity between completely different target types and completely custom groupings - independent of e.g. organization of tracks into folders.

Inverse target value (on only)

Variation of Inverse target value that applies the inverse only when the target value is > 0%.

Inverse target value (off only)

Variation of Inverse target value that applies the inverse only when the target value is 0%.

Feedback type controls

Feedback ←

specifies whether to send simple numeric, simple text or script-generated feedback values to the source

See Feedback type.

Feedback style menu (…)

The … button provides options to change the *feedback style*. At the moment, it's all about setting colors.

If you use Dynamic feedback: Lua script, changes made here don't have any effect because you are supposed to provide style properties as part of the Luau script result (which is much more flexible).

Color / Background color

With this you can define the color and background color of the displayed text. Of course this will only work if the source supports it!

<Default color>

Chooses the default color, that is the one which is preferred for the corresponding controller and display type.

<Pick color…>

Opens a color picker so you can choose the color of your choice.

Property name

Maybe you don't want a fixed color but a dynamic one that changes whenever your target changes. Choose one of the properties to make that happen. Do a full-text search in the reference to learn about the meaning of the property.

Source Min/Max controls

Control →

sets the observed range of absolute control values coming from the source

Feedback ←

sets the minimum and maximum feedback value

Doesn't have an effect on relative control values.

Depending on the context, this can have different effects:

Element	Direction	Effect
Momentary button	Control →	If min > 0 and Out-of-range behavior menu is Ignore, button releases are ignored. Because this also affects Feedback ←, it's usually better to use the Button filter menu instead!
Velocity-sensitive button (key, pad, …)	Control →	Defines the observed velocity range, for example to react to only the lower velocity layer of a key press.

Element	Direction	Effect
Range element (fader, knob, …)	Control →	Defines the observed value range. For example to react only to the upper half of a fader.
LED	Feedback ←	On many controllers which support colored LEDs, **Min** sets the **off** color and **Max** sets the **on** color.
Value indicator (LED ring, motor fader, …)	Feedback ←	Sets the lowest/highest indicated value.

By restricting that range, you basically tell ReaLearn to or only the lower velocity layer of a key press.

This range also determines the minimum and maximum Feedback value.

Out-of-range behavior menu

Control →

determines what to do if the absolute control value coming from the source is not within source min/max

Feedback ←

determines what to do if the feedback value is not within target min/max

See Source Min/Max controls and Target Min/Max controls.

There are the following options:

	Control →	Feedback ←
Min or max	If the source value is < *Source Min*, ReaLearn will behave as if *Source Min* was received (or 0% if *Source Min* = *Source Max*). If the source value is > *Source Max*, ReaLearn will behave as if *Source Max* was received (or 100% if *Source Min* = *Source Max*).	If the target value is < *Target Min*, ReaLearn will behave as if *Target Min* was detected (or 0% if *Target Min* = *Target Max*). If the target value is > *Target Max*, ReaLearn will behave as if *Target Max* was detected (or 100% if *Target Min* = *Target Max*).
Min	ReaLearn will behave as if *Source Min* was received (or 0% if *Source Min* = *Source Max*).	ReaLearn will behave as if *Target Min* was detected (or 0% if *Target Min* = *Target Max*). Useful for getting radio-button-like feedback.

	Control →	Feedback ←
Ignore	Target value won't be touched.	No feedback will be sent.

Mode ("Absolute mode") menu

Control →

specifies how incoming absolute control values are interpreted and handled

Not all modes make sense at all times! It mostly depends on the character of the source. If a mode doesn't make sense given the current source, it will be marked as `NOT APPLICABLE`. In this case, you should choose another mode or change the source.

Normal

Takes and optionally transforms absolute source control values *the normal way*. *Normal* means that the current target value is irrelevant and the target will just be set to whatever absolute control value is coming in (potentially transformed).

Incremental button

With this you can "go relative" with buttons instead of encoders in a "previous/next fashion".

Let's assume you use the *MIDI Note velocity* and select *Incremental button* mode. Then it works like this: Each time you press the key, the target value will increase, according to the mode's settings. You can even make the amount of change velocity-sensitive! If you want the target value to decrease, just check the Reverse checkbox.

Toggle button

Toggle button mode is used to toggle a target between on and off states. It only makes sense for momentary buttons (which fire a value > 0 on each press).

Here's how it works in detail:

- If the current target value is within the first half of the target min/max range, it's considered as *off* and will therefore be switched *on* (set to *target max*). If it's within the second half, it's considered as *on* and will therefore be switched *off* (set to *target min*).
- It works a bit differently if *target min* and *target max* have the same value (which is a common technique to set the target to a specific value on the press of a button). Instead of toggling between *target min* and *target max*, this mode now toggles between this specific value (= *target min* = *target max*) and 0%. This is useful whenever you have a set of buttons each of which sets the same target to a different value, and you want them to toggle between the specified value and an initial value (0%).

  This mode is not supported for controller mappings that have a virtual target.

Sometimes the controller itself provides a toggle mode for buttons. **Don't use it!**

Always set up your controller buttons to work in momentary mode! It's

impossible for the controller to know which state (on/off) a target currently has. Therefore, if you use the controller's built-in toggle function, it's quite likely that it gets out of sync with the actual target state at some point.

ReaLearn's own toggle mode has a clear advantage here.

Make relative

This converts incoming absolute fader/knob movements into relative adjustments of the target value. It somewhat resembles takeover mode Parallel but has important differences:

- It's guaranteed that a full fader/knob swipe from 0% to 100% always results in a swipe over the full target range (assuming the target was at 0% initially).
- It doesn't need to know the current target value. Which means it also works for mappings with virtual targets.

Performance control

This mode emulates the behavior of a typical soft synth modulation matrix mapping: It uses the target value that has been set in REAPER (not via this ReaLearn mapping) as an offset and starts changing it from there.

Round target value checkbox

Control →

rounds target values to the nearest integer

Only a few targets support that, such as :targets/project/set-tempo.pdf.

Takeover mode menu

Control →

defines how to deal with potential control value jumps

If you are not using motorized faders, absolute mode is inherently prone to parameter jumps. A parameter jump occurs if you touch a control element (e.g. fader) whose position in no way reflects the current target value. This can result in audible jumps because the value is changed abruptly instead of continuously. You can deal with this by setting the right takeover mode.

ReaLearn provides multiple takeover modes that decide how to deal with situations when a target parameter jump would occur.

Off

The default setting: Jumps allowed.

Pick up

This is the same as *Soft takeover* in REAPER's built-in MIDI learn. It prevents jumps by not changing the target value until your control element reaches it.

In certain cases, this mode can cause the target value to get stuck. This happens with faders/knobs that cause jumps themselves when moved very rapidly. If you don't like that, you

might want to try Pick up (tolerant).

Pick up (tolerant)

This is like Pick up but makes extra sure that the target value doesn't get stuck.

However, unlike Pick up, this mode will jump if you cause a jump on your controller! Imagine using a touch strip. This kind of control element allows you to jump to arbitrary values at any time. Tolerant mode will not prevent this kind of jumps!

Long time no see

This is similar to Pick up with the difference that the current target value will gradually "come your way". This results in seamless and fast reunification of control and target value, but it can feel weird because the target value can temporarily move in the opposite direction of the fader movement. In older ReaLearn versions this was called "Slowly approach if jump too big".

Parallel

With this mode, the target will simply follow your fader moves, in exactly the same tempo - without any scaling. Reunification only happens when both control and target value meet at the "borders".

Catch up

This mode is sometimes called "Proportional" or "Value scaling" mode. It's like "Parallel" mode but the target value is allowed to move slower than the control value - hence the control can catch up (converge) faster.

Control transformation (EEL) field

Control →

applies highly customizable transformations or filtering to incoming absolute control values

This feature allows you to write a formula that transforms incoming control values. While very powerful because it allows for arbitrary transformations (velocity curves, random values - you name it), it's not everybody's cup of tea to write something like that. The formula must be written in the language EEL2. Some REAPER power users might be familiar with it because REAPER's JSFX uses the same language.

Luckily, ReaLearn has a fancy editor which visualizes the formula and has some predefined templates built-in (available on Windows and macOS only at the moment). Press the "**...**" button to open the editor. Code changes are applied immediately.

The most simple formula is `y = x`, which means there will be no transformation at all. `y = x / 2` means that incoming control values will be halved. You get the idea: `y` represents the desired target control value (= output value) and `x` the incoming source control value (= input value). Both are 64-bit floating point numbers between 0.0 (0%) and 1.0 (100%).

The script can be much more complicated than the mentioned examples and make use of all built-in EEL2 language features. The important thing is to assign the desired value to `y` at some point.

The following variables/functions are available in the formula:

`y`

Initially contains the *current* target value. You can use that value in order to calculate the new value. With this, you can essentially craft your own relative mode!

`y_last`

This contains the last value of the target before it was affected by this particular mapping.

Allows you to come up with a performance control mode typical for synth parameter mappings, just like the built-in Performance control mode but more customizable. Try this for example: `y = y_last + x * (1 - y_last)`

`rel_time`

This contains the number of milliseconds since this mapping has last been triggered with a control message coming from the source.

As soon as you use this and a control message comes in, ReaLearn will start invoking your formula *repeatedly*! That means, this variable is your entrance ticket to smooth transitions and continuous parameter modulation.

A few examples:

- Smooth transition from current value to control value: `rel_time; y = abs(x - y) < 0.05 ? stop : y + 0.1 * (x - y)`
- Sinus LFO: `y = (sin(rel_time / 500) + 1) / 2`
- Linear transition to control value (1 second): `y = abs(x - y) < 0.05 ? stop : x * min(rel_time / 500, 1)`
- 2 seconds chaos: `y = rel_time < 2000 ? rand(1) : stop`
- Setting a value with delay: `y = rel_time < 2000 ? none : stop(0.5)`

`stop` **and** `stop(…)`

In combination with `rel_time`, this stops repeated invocation of the formula until the mapping is triggered again.

Good for building transitions with a defined end.

Stopping the invocation at some point is also important if the same parameter should be controlled by other mappings as well. Otherwise, if multiple mappings continuously change the target parameter, only the last one wins.

This also exists as a function, which lets you do both, returning a target value **and** stopping the transition. Pass the desired value in the parentheses, e.g. `stop(0.5)`.

`none`

Usually, each repeated (see `rel_time`) invocation always results in a target invocation (unless the target is not retriggerable and already has the desired value). Sometimes this is not desired. In this case, one can return `none`, in which case the target will not be touched.

Good for transitions that are not continuous, especially if other mappings want to control the parameter as well from time to time.

ReaLearn's control processing order is like this:

1. Apply source interval
2. Apply transformation
3. Apply reverse
4. Apply target interval
5. Apply rounding

Step size Min/Max controls

Control →

specifies by which amount to increase the target value when an increment or decrement is received

When you deal with relative adjustments of target values in terms of increments/decrements, then you have great flexibility because you can influence the *amount* of those increments/decrements. This is done via the *Step size* setting, which is available for all *continuous* targets.

Step size Min

Specifies how much to increase/decrease the target value when an increment/decrement is received.

Depending on the context, this can have different effects:

Element	Direction	Effect
Incremental button	Control →	Sets the target value change amount when button pressed
Incremental velocity-sensitive button (key, pad, …)	Control →	Sets the target value change amount when button pressed with the lowest velocity
Rotary endless encoder	Control →	Sets the target value change amount for an incoming non-accelerated increment/decrement
Rotary endless encoder with Make absolute	Control →	Sets the amount added/subtracted to calculate the simulated absolute value from an incoming non-accelerated increment/decrement

Step size Max

Sets the maximum amount by which to increase/decrease the target value with one interaction. If you set this to the same value as *Min*, encoder acceleration or changes in velocity will have no effect on the incrementation/decrementation amount. If you set it to 100%, the effect is

maximized.

Depending on the context, this can have different effects:

Element	Direction	Effect
Incremental velocity-sensitive button (key, pad, …)	Control →	Sets the target value change amount when button pressed with the highest velocity
Rotary endless encoder	Control →	If the hardware encoder supports acceleration, this sets the target value change amount for the most accelerated increment/decrement

Speed Min/Max controls

Control →

specifies how often to increase or decrease the target value when an increment or decrement is received

When you choose a target with a Discrete value range, the Step size Min/Max controls label will change into *Speed*. If a target is discrete, it cannot have arbitrarily small step sizes. It rather has one predefined atomic step size. Allowing arbitrary step size adjustment wouldn't make sense. That's why *Speed* allows you to *multiply* (positive numbers) or *"divide"* (negative numbers) value increments with a factor instead. Negative numbers are most useful for rotary encoders because they will essentially lower their sensitivity. Virtual targets are always discrete.

Example 4. Speed example

Let's assume you selected the discrete Target "FX: Browse presets", which is considered discrete because an FX with for example 5 presets has 6 well-defined possible values (including the <no preset> option), there's nothing inbetween. And let's also assume that you have a controller like Midi Fighter Twister whose rotary encoders don't support built-in acceleration.

Now you slightly move an encoder clock-wise and your controller sends an increment +1. If the *Speed Min* slider was at 1 (default), this will just navigate to the next preset (+1). If the *Speed Min* slider was at 2, this will jump to the 2nd-next preset (+2). And so on.

Remarks:

- There are FX plug-ins out there which report their parameter as discrete with an insanely small step size (e.g. some Native Instrument plug-ins). This kind of defeats the purpose of discrete parameters and one can argue that those parameters should actually be continuous. In such a case, moving your rotary encoder might need *a lot* of turning even if you set *Speed* to the apparent maximum of 100! In this case you will be happy to know that the text field next to the slider allows you to enter values higher than 100.

- You can set the "Speed" slider to a negative value, e.g. -2. This is the opposite. It means you need to make your encoder send 2 increments in order to move to the next preset. Or -5: You need to make your encoder send 5 increments to move to the next preset. This is like slowing down the encoder movement.

Encoder filter menu

Control →

allows reacting to either clockwise or counter-clockwise encoder turns

For example, if you want to invoke one action on clockwise movement and another one on counter-clockwise movement. Or if you want to use different step sizes for different movements.

Increment & decrement

ReaLearn will process both increments and decrements.

Increment only

ReaLearn will ignore decrements.

Decrement only

ReaLearn will ignore increments.

Wrap checkbox

Control →

makes the target value jump back to target min when target max is reached

If unchecked, the target value will not change anymore if there's an incoming decrement but the target already reached its minimum value. If checked, the target value will jump to its maximum value instead. It works analogously if there's an incoming increment and the target already reached its maximum value.

If this flag is enabled for controller mappings which have a virtual target, every main mapping controlled by that virtual control element will wrap - even if the main mapping itself doesn't have Wrap checkbox enabled.

Make absolute

Control →

emulates an absolute control element with a relative encoder or with -/+ buttons (= incremental buttons)

This is useful if you have configured your controller to be relative all the way (which is good!) but you want to use a control transformation EEL formula - which is not possible if you change the target with relative increments. It works by keeping an internal absolute value, incrementing or decrementing it accordingly and then processing it just like normal absolute control values.

By checking this box:

- You lose the possibility to be perfectly free of parameter jumps (but you can try to mitigate that

loss by using the jump settings).

- You gain support for control-direction EEL transformation, non-continuous target value sequences and source range.
- You can still use some of the relative-only features: Step size and rotate!

Fire mode menu

Control →

allows differentiating between long-press, short-press, and double-press actions

Normally, when a button gets pressed, it controls the target immediately. However, by using this dropdown and by changing the values below it, you can change this behavior. This dropdown provides different fire modes that decide how exactly ReaLearn should cope with button presses.

Fire on press (or release if > 0 ms)

This mode is essential in order to be able to distinguish between different press durations.

- **Min** and **Max** decide how long a button needs to be pressed to have an effect.
- By default, both min and max will be at 0 ms, which means that the duration doesn't matter and both press (> 0%) and release (0%) will be instantly forwarded. If you change *Min* to e.g. 1000 ms and *Max* to 5000 ms, it will behave as follows:
- If you press the control element and instantly release it, nothing will happen.
- If you press the control element, wait for a maximum of 5 seconds and then release it, the control value of the press (> 0%) will be forwarded.
- It will never forward the control value of a release (0%), so this is probably only useful for targets with trigger character.
- The main use case of this setting is to assign multiple functions to one control element, depending on how long it has been pressed. For this, use settings like the following:
- Short press: 0 ms - 250 ms
- Long press: 250 ms - 5000 ms

Fire after timeout

This mode is more "satisfying" because it will let ReaLearn "fire" immediately once a certain time has passed since the press of the button. However, obviously it doesn't have the concept of a "Maximum" press duration, so it can't be used to execute different things depending on different press durations (or only as the last part in the press duration chain, so to say).

Timeout

Sets the timeout in milliseconds. If this is zero, everything will behave as usual.

Fire after timeout, keep firing (turbo)

Welcome to turbo mode. It will keep hitting your target (always with the initial button press velocity) at a specific rate. Optionally with an initial delay. Epic!

Timeout

This is the initial delay before anything happens. Can be zero, then turbo stage is entered instantly on press.

Rate

This is how frequently the target will be hit once the timeout has passed. In practice, it won't happen more frequently than once every 30 ms (REAPER's main thread loop frequency).

Fire on double press

This reacts to double presses of a button (analog to double-clicks with the mouse).

Fire after single press (if hold < Max ms)

If you want to do something in response to a double press, chances are that you want to do something *else* in response to just a single press. The *Normal* fire mode will fire no matter what! That's why there's an additional *Single press* mode that will not respond to double presses. The response happens *slightly* delayed - because ReaLearn needs to wait a bit to see if it's going to be a double press or not.

Max

With this, it's even possible to distinguish between single, double *and* long press. In order to do that, you must set the *Max* value of the *Single press* mapping to a value that is lower than the *Timeout* value of your *After timeout* mapping. That way you can use one button for 3 different actions!

- Mapping 1 "Single press" with Max = 499ms
- Mapping 2 "Double press"
- Mapping 3 "After timeout" with Timeout = 500ms

Button filter menu

Control →

allows reacting to either button press or button release events

Press & release

ReaLearn will process both button presses (control value = 0%) and button releases (control value > 0%). This is the default.

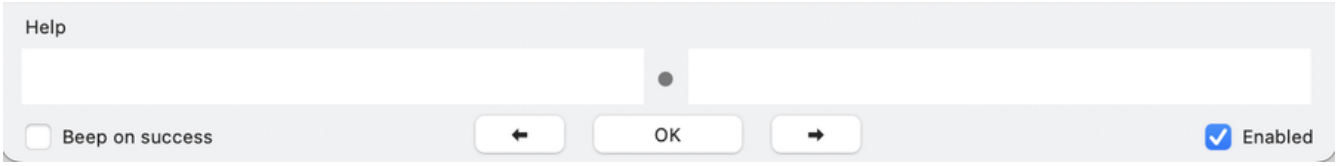
Press only

Makes ReaLearn ignore the release of the button. The same thing can be achieved by setting *Source Min* to 1. However, doing so would also affect the feedback direction, which is often undesirable because it will mess with the button LED color or on/off state.

Release only

Makes ReaLearn ignore the press of the button (just processing its release). Rare, but possible.

Bottom section



The section at the bottom has the following functions:

- To provide context-sensitive help for the glue section
- To provide control information, feedback information and error reporting
- To provide navigation and general mapping control

Help

Context-sensitive help for the glue section. Whenever you touch a setting in the glue section, some text will appear which explains what this element does, both for the *control* and for the *feedback* direction (if applicable).

Activity info area

The left text area shows information about how an incoming control value was handled and possible target control errors.

If the target supports MIDI real-time control and the source is a MIDI source, this currently only works if "Log target control" is enabled (see Logging Menu).

The right text area shows information about which feedback values are sent from the glue section to the source.

Beep on success checkbox

Makes the mapping play a sound whenever the target has been invoked successfully. Nice for trigger-like targets such as Target "ReaLearn: Take mapping snapshot" for which there's no other good way to know if it worked.

Previous button

Allows you to jump to the previous mapping. Considers only mappings that are currently visible in the mapping rows panel.

Next button

Allows you to jump to the next mapping. Considers only mappings that are currently visible in the mapping rows panel.

Enabled checkbox

Enables or disables the mapping as a whole.

Advanced settings dialog

This dialog allows access to advanced configuration by entering text in the YAML configuration language. It can be opened using the Advanced settings button of the mapping panel.

See Advanced settings via YAML to learn why this feature exists and why YAML.

Expert level!

The YAML language

This is not a programming language, so you can't write loops, conditions or anything like that. Instead, think of it as a language for writing configuration. Do you know INI files? REAPER uses INI files to save configuration. YAML is a bit like that, just much more expressive because it allows you to not only express flat key-value pairs (e.g. `edit_fontsize=29`) but also deeply nested configuration data and lists.

YAML is indentation-sensitive

Indentation matters! The bright side of this is that it always looks clean. The dark side is that ReaLearn will refuse to save your settings if you messed up the indentation.

Therefore: Be consistent with your indentation (e.g. use always an indentation of 2 spaces for nesting) and have an utmost attention to detail when doing copy and paste from the examples in this section!

Verbatim text is not saved

When you close the text editor and ReaLearn saves your advanced settings as part of the mapping, it will not save the text that you have entered *verbatim*. It will save a structural representation of what you entered. Plus, it will strip comments!. That means if you open the advanced settings again, your text could look a bit different, in particular it can have a different formatting. But don't worry, it *means* exactly the same to ReaLearn.

Supported configuration properties

In this section you will find examples that cover all currently supported configuration properties. You can copy and paste the stuff you need to the text editor, remove the parts that you don't need and adjust the rest. Comments (lines starting with `#`) will be removed automatically.

Mapping lifecycle actions

ReaLearn allows you to define MIDI messages to be sent to the output whenever a mapping turns active or inactive. See Mapping activation state.

Example use cases:

- Accessing device-specific features via system-exclusive MIDI messages.

- Choosing a different LED color/style depending on the active mapping.
- Initializing a sys-ex-controllable display with some mapping-specific text (more difficult).

These are the available configuration properties:

```
# Contains stuff to be done whenever this mapping becomes active.
on_activate:
  # A list of MIDI messages to be sent to the output when this mapping becomes active.
  #
  # At the moment, only messages of type "raw" are supported. Although this covers all
possible types
  # of MIDI messages, it's a bit hard to express e.g. simple NOTE ON or CC messages
with this notation.
  # In particular, you would need to know how MIDI messages are presented as byte
sequences. Future ReaLearn
  # versions will provide more convenient ways to describe simple MIDI messages.
  send_midi_feedback:
    # This is an example of a system-exclusive message ("SysEx"). It's usually
expressed in hexadecimal string
    # notation. Make sure to include the leading F0 and trailing F7, which is the
begin and end marker of all
    # system-exclusive messages!
    - raw: F0 00 20 6B 7F 42 02 00 10 77 01 F7
    # Instead of above hexadecimal string notation, you could also use an array of
decimal numbers to describe a raw
    # message. The following is a NOTE ON of note 74 on channel 1 with velocity 100.
    - raw:
        # NOTE ON on channel 1
        - 144
        # Note number 74
        - 74
        # Note velocity 100
        - 100

# Contains stuff to be done whenever this mapping becomes inactive.
on_deactivate:
  # A list of MIDI messages to be sent to the output when this mapping becomes
inactive.
  send_midi_feedback:
    # Supports exactly the same kinds of messages as described above in "on_activate".
    - raw: F0 00 20 6B 7F 42 02 00 10 77 14 F7
```

Please remember that YAML comments (e.g. `# The following line does this and that`) *will not be saved*! In case you want to explain something, you need to write it as YAML property, such as in the following example:

```
comment: "The following configuration makes the rightmost pad of the MiniLab mkII
light up in red color."
on_activate:
```

```
send_midi_feedback:
  - raw: F0 00 20 6B 7F 42 02 00 10 77 01 F7
```

ReaLearn will ignore any unknown properties.

If you use input MIDI: <FX input> and find that MIDI lifecycle messages aren't sent, no matter what, make sure "Send feedback only if track armed" is disabled (see Unit options)!

Disabling the complete ReaLearn instance will cause all mappings in all units of that instance to deactivate. However, sending MIDI messages on deactivation in this case will only work if the output is a device! If it is MIDI: <FX output>, it will not send anything because REAPER will not give that ReaLearn instance any chance to output MIDI messages once it's disabled. Instead, the MIDI message will queue up and be sent once you enable that instance again … which is probably not what you want.

Open in text editor button (Windows and Linux only)

Opens the settings in the system text editor or whatever program is associated with YAML files. It depends on your system setup if this works or not. If it does and if your text editor is good, this can make editing larger YAML snippets more convenient (e.g. by providing syntax highlighting).

As soon as you save the file and close the editor, the text will automatically appear in the "Advanced settings" text area.

Help button

Will open an online version of the user guide section that describes the available configuration properties.

Sources

ReaLearn supports the following Source types.

Source "None"

A special kind of source that will never emit any events. It's intended to be used on mappings which are not supposed to be controlled directly but only via Group interaction.

MIDI sources

Most types in the MIDI category have the following UI elements in common.

Channel menu

Optionally restricts this source to messages from a certain MIDI channel. Only available for sources that emit MIDI channel messages.

The remaining UI elements in this section depend on the chosen source type.

Source "CC value"

This source reacts to incoming MIDI control-change messages.

CC menu

Optionally restricts this source to messages with a certain MIDI control-change controller number.

Character menu

See MIDI source character.

14-bit values checkbox

If unchecked, this source reacts to MIDI control-change messages with 7-bit resolution (usually the case). If checked, it reacts to MIDI control-change messages with 14-bit resolution. This is not so common but sometimes used by controllers with high-precision faders.

Source "Note velocity"

This source reacts to incoming MIDI note-on and note-off messages. The higher the velocity of the incoming note-on message, the higher the absolute control value. Note-off messages are always translated to 0%, even if there's a note-off velocity.

Note menu

Optionally restricts this source to messages with a certain note number (note numbers represent keys on the MIDI keyboard, e.g. 60 corresponds to C4).

Source "Note number"

This source reacts to incoming MIDI note-on messages. The higher the note number (= key on a MIDI keyboard), the higher the absolute control value.

This essentially turns your MIDI keyboard into a "huge fader" with the advantage that you can jump to any value at any time.

Source "Pitch wheel"

This source reacts to incoming MIDI pitch-bend change messages. The higher the pitch-wheel position, the higher the absolute control value. The center position corresponds to an absolute control value of 50%.

Source "Channel after touch"

This source reacts to incoming MIDI channel-pressure messages. The higher the pressure, the higher the absolute control value.

Source "Program change"

This source reacts to a range of incoming MIDI program-change messages. The higher the program number, the higher the absolute control value.

Source "(N)RPN value"

This source reacts to incoming non-registered (NRPN) or registered (RPN) MIDI parameter-number messages. The higher the emitted value, the higher the absolute control value.

(N)RPN messages are not widely used. If they are, then mostly to take advantage of their ability to transmit 14-bit values (up to 16384 different values instead of only 128), resulting in a higher resolution.

RPN checkbox

If unchecked, this source reacts to unregistered parameter-number messages (NRPN). If checked, it reacts to registered ones (RPN).

Number field

The number of the registered or unregistered parameter-number message. This is a value between 0 and 16383.

14-bit values checkbox

If unchecked, this source reacts to (N)RPN messages with 7-bit resolution, including increment/decrement messages.

If checked, it reacts to those with 14-bit resolution. In practice, this if often checked.

Character menu

See MIDI source character.

Source "Polyphonic after touch"

This source reacts to incoming MIDI polyphonic-key-pressure messages. The higher the pressure, the higher the absolute control value.

Note menu

Optionally restricts this source to messages with a certain note number.

Source "MIDI clock tempo"

This source reacts to incoming MIDI clock (MTC) tempo messages. These are metronome-beat-like messages which can be regularly transmitted by some DAWs and MIDI devices. The frequency with which this message is sent dictates the tempo.

The higher the calculated tempo, the higher the absolute control value. A tempo of 1 bpm will be translated to a control value of 0%, a tempo of 960 bpm to 100% (this corresponds to REAPER's supported tempo range).

This source can be used in combination with the Target "Project: Set tempo" to obtain a "poor man's" tempo synchronization.

Be aware: MIDI clock naturally suffers from certain inaccuracies and latencies - that's an issue inherent to the nature of the MIDI clock protocol itself. E.g. it's not really suitable if you need super accurate and instant tempo synchronization. Additionally, ReaLearn's algorithm for calculating the tempo could probably be improved (that's why this source is marked as experimental).

Source "MIDI clock transport"

This source reacts to incoming MIDI clock (MTC) transport messages. These are simple start, continue and stop messages which can be sent by some DAWs and MIDI devices.

Message menu

The specific transport message to which this source should react.

Source "Raw MIDI / SysEx"

This source primarily deals with system-exclusive MIDI messages. It supports both control and feedback direction!

Pattern field

Pattern describing the raw MIDI message. See Raw MIDI pattern.

Source "MIDI Script"

This source is feedback-only and exists for enabling more complex feedback use cases such as controlling LCDs that are not yet supported by the Source "Display". It lets you write an EEL or Luau script that will be executed whenever ReaLearn "feels" like it needs to send some feedback to the MIDI device.

Kind menu

Whether to use the EEL or Luau language.

Script field

The script. Is disabled if the script contains more than one line.

See MIDI source script for details.

More button (...)

Opens the script in a separate window (for multi-line scripts).

Prefer the Source "Display" over this one whenever possible. It's easier to use.

Source "Display"

This is a feedback-only source used to display text on MIDI-controllable hardware displays (LCDs, OLED displays, 7-segment displays, ...).

Protocol menu

Lets you choose the display protocol, which tells ReaLearn how it should communicate with the hardware display and which options it supports.

Mackie LCD

Use this for MCU-compatible LCDs. Depending on your particular control surface, there can be up to 8 LCDs, each of which has up to 2 lines.

Mackie XT LCD

Use this to control the displays of MCU XT devices (= control surface extenders, which provide additional faders and displays).

X-Touch Mackie LCD

Like *Mackie LCD* but also supports colors on certain X-Touch devices.

X-Touch Mackie XT LCD

Like *Mackie LCD XT* but also supports colors on certain X-Touch devices.

Mackie 7-segment display

Use this for MCU-compatible 7-segment displays (you know, the ones which only show digits).

There's usually one small assignment display and a larger one for showing the time code.

SiniCon E24

Use this with the SiniCon E24 controller.

Launchpad Pro - Scrolling text

Displays looped scrolling text on a Novation Launchpad Pro. Only seems to work if you set *Output* to `MIDIOUT2 (Launchpad Pro)`.

Studiologic SL Keyboard display

Displays text on the display of Studiologic SL keyboards (tested with SL88).

Display menu

Choose the particular display or display portion to which you want to send text.

Line menu

Choose the line number.

For controllers with multiple displays and lines, ReaLearn allows you to spread your text over all available displays and lines. This is great if you need to display a lot of text but one display doesn't provide enough space. But be aware: Feedback relay doesn't work nicely anymore if you make use of this feature.

If you want to know how to define which text shall be sent to the displays, please see textual feedback in the Glue section.

Source "Specific program change"

This source reacts to MIDI program-change messages with a specific program. This is a trigger-only source, that means it always fires 100% (whenever the program number corresponds to the configured one).

Source "OSC"

OSC sources allow configuration of the following aspects:

Address field

This needs to correspond exactly to the address of the corresponding control element on your OSC device.

Example: `/1/fader1`

You don't need to figure that out yourself, just use the **[Learn]** button.

Argument section

Each OSC message consists of an arbitrary number of arguments. In most cases, e.g. with faders, knobs or buttons, it's just one argument. X/Y controls often send 2 arguments, one for each axis. There are rare cases in which messages have even more arguments.

Argument number menu

The first dropdown menu allows you to choose the number of the argument that ReaLearn should look at and process. `1` denotes the first argument, `2` the second one, and so on.

Argument type menu

The second dropdown menu lets you choose the argument type which ReaLearn should use to construct a proper feedback message.

This is usually the same type as the one used for control direction. For control direction, choosing an explicit type is irrelevant because ReaLearn handles whatever type arrives automatically in the best possible way.

If you use *Learn*, the type is filled automatically. * The value to be sent will be derived from the type (see Feedback arguments field):

Type	Property
`Float`	`value.float`
`Double`	`value.double`
`Int`	`value.int`
`Long`	`value.long`
`Bool`	`value.bool`
`Nil`	`nil`
`Inf`	`inf`
`String`	`value.string`
`Color`	`style.color`

If you want more control over what feedback values are sent, use the Feedback arguments field field.

Range field

Values of argument types *Float* and *Double* are by default interpreted as decimal values between 0.0 and 1.0. You can change that by entering a different value range here. Even negative numbers are allowed.

Customizing the value range is especially important for argument types *Int* and *Long* because they don't have a standard value range.

Is relative checkbox

Some messages transmitted by OSC devices are meant to be interpreted as relative increments/decrements instead of absolute values, e.g. jog wheels. When you enable this checkbox, ReaLearn will treat each received *1* value as an increment and *0* value a decrement.

Feedback arguments field

Allows you to modify the OSC feedback arguments expression.

Source "Keyboard"

This source reacts to pressing or releasing a key on your computer keyboard. It emits a value of 100% when the key is pressed and 0% when released.

In order to set the key, simply click the **[Learn]** button and press the key of your choice.

In addition to the key label, ReaLearn might show some warnings regarding the portability of your keystroke. This helps you to avoid keyboard shortcuts that don't reliably work cross-platform (in other operating systems) or on other keyboard layouts. You can ignore portability warnings if you use just this operating system and don't plan to share your keyboard presets with other users.

- This only works if Computer keyboard is enabled in the Input menu.
- If you hold a key, it will not keep firing. This is by design! Use Fire after timeout, keep firing (turbo) instead.
- Key combinations are not supported. This is by design! Use Conditional activation instead.

REAPER sources

Source "MIDI device changes"

This source emits a value of 100% whenever any MIDI device is connected and 0% whenever any MIDI device is disconnected. You can map this to the REAPER action "Reset all MIDI devices" to achieve true plug and play of MIDI devices (provided the corresponding device has been enabled at least once in REAPER's MIDI device preferences).

Source "ReaLearn unit start"

This source fires (emits a value of 100%) when ReaLearn starts. It can be used to execute an actions or restore certain states on REAPER startup or project load.

Source "Timer"

This source fires (emits a value of 100%) repeatedly every *n* milliseconds.

Source "ReaLearn parameter"

This source fires whenever a selected Compartment parameter changes its value.

One of many ways to use this is to create macro parameters which control multiple parameters of multiple other plug-ins.

This is one of the sources that can't participate in rendering. So it's important to write down automation **before** rendering.

Source "Speech"

This source works for feedback only. It uses the native Windows or macOS text-to-speech engine to speak out any feedback value.

Source "Virtual"

See Virtual source.

Type menu

See Virtual control element type.

ID section

See Virtual control element ID.

Pick menu

The convenient picker provides IDs from standardized virtual control schemes:

ID field

Lets you enter the ID manually.

Targets

Global targets

Target "Global: Last touched"

This will control whatever target has been last touched in REAPER. It's similar to the built-in REAPER action "Adjust last touched FX parameter" but provides the following benefits:

1. It's applicable to all ReaLearn targets that are learnable, not just FX parameters.
2. It offers feedback.
3. It can distinguish between parameter modifications caused by ReaLearn (i.e. hardware control) and those caused in other ways (e.g. via mouse).

Pick button

This opens a window that lets you pick all considered target types and types of invocations (only macOS and Windows so far). Last-touched targets not checked in this window will be ignored.

Target "Global: Mouse"

This will control the mouse.

Action menu

Move cursor to

Moves the mouse cursor on the given axis in an absolute manner. This is a good choice for absolute mouse movement, that is, if you want to position the mouse cursor to a specific screen position. Although it's also possible to move the mouse cursor relatively with this action by controlling the target with relative messages, it's usually better to use *Move cursor by* instead.

Move cursor by

Moves the mouse cursor on the given axis in a relative manner. This is a good choice if you want to move the cursor e.g. up a bit, starting from its current position. This only works with relative control elements such as encoders or features such as Make relative.

Press or release button

Presses or releases a certain mouse button, depending on the incoming control value (0% = release, anything else = press).

Turn scroll wheel

Simulates the scroll wheel.

Axis menu

Determines the direction of movement or scrolling.

X (horizontal)

Horizontal movement or scrolling

Y (vertical)

Vertical movement or scrolling

Button menu

Determines which mouse button to use.

One popular use of this target is to adjust the FX parameter under the mouse cursor. For this, it's usually best to use action Turn scroll wheel and Y (vertical).

You can unfold the magic of this target by combining multiple mappings. E.g. one can simulate mouse dragging by using one mapping to press/release the left button and another mapping to move the cursor. This example project contains multiple examples (one per group).

Feedback for this target is not fully implemented.

Target "Global: Set automation mode override"

Sets the global automation mode override to the desired value if the incoming control value is greater than 0%, otherwise removes the override.

Behavior menu

Lets you decide between not overriding anything, bypassing all envelopes or overriding with a specific automation mode.

Mode menu

Here you can pick the desired automation mode if *Behavior* is *Override*.

Project targets

Target "Project: Any on (solo/mute/...)"

This target is most useful in feedback direction. Map it to some LED on your controller and the LED will light up if at least one of the tracks in your project is e.g. mute (depending on the track parameter in question).

If the control element is also a button, pressing the button will e.g. unmute all tracks in your project.

Parameter menu

The track parameter in question.

Target "Project: Invoke REAPER action"

Triggers or sets the value of a particular REAPER action in the main section.

Section menu

Specifies in which context the action is going to be invoked.

Main
: Invokes a main action.

Active MIDI editor
: Invokes a MIDI editor action, applied to the currently active MIDI editor.

Active MIDI event list editor
: Invokes a MIDI event list action, applied to the currently active MIDI editor.

Media explorer
: Invokes a media explorer action.

Invocation type

Specifies *how* the picked action is going to be controlled.

Trigger
: Invokes the action with the incoming absolute control value, but only if it's greater than 0%. Most suitable for simple trigger-like actions that neither have an on/off state nor are annotated with "(MIDI CC/OSC only)" or similar.

Absolute 14-bit
: Invokes the action with the incoming absolute control value, even if it's 0%. Most suitable for actions which either have an on/off state or are annotated with "(MIDI CC/OSC only)" or similar. The resolution of the invocation is 14-bit, no matter what's the resolution of your control element).

Absolute 7-bit
: Just like the previous invocation mode but uses 7-bit resolution. Might be necessary for actions provided by 3rd-party extensions which don't interpret 14-bit control values correctly. In all other circumstances, 14-bit is probably the better default choice.

Relative
: Invokes the action with the incoming relative control value (absolute ones are ignored). Only works for actions that are annotated with ("MIDI CC relative only") or similar.

Pick! button

Opens REAPER's action dialog so you can select the desired action.

With track checkbox

Allows you to choose a track which ReaLearn will select before executing the action. This makes it possible to combine ReaLearn's flexible track selection capabilities with the plethora of REAPER actions that work on the currently selected track.

Limitations

The particular action decides if toggling/feedback works completely, has limitations or is not possible at all. There are multiple types of actions so it's not possible to settle with one invocation type and be done with it. The types of actions can roughly be divided into:

Actions that take care of toggling themselves *and* report on/off state

- Example: "25. Track: Toggle record arm for track 01"
- If you want toggle behavior, you have 2 options:
  - a) Set Invoke to "Absolute" and Mode to "Toggle button" (preferred).
  - b) Set Invoke to "Trigger" and Mode to "Normal".
- Feedback is completely supported.

Actions that take care of toggling themselves but *don't* report on/off state

- Example: "40175. Item properties: Toggle mute"
- Toggle behavior is achieved as described in (1) but support for toggling and feedback has limitations (explained in (4)).

Actions that don't take care of toggling themselves ("trigger only")

- Example: "1007. Transport: Play"
- There's no way to make such an action toggle because the action is not designed to do so.
- If the action reports an on/off state, feedback is completely supported though, otherwise not at all!

Actions that have a complete range of values as state

- Example: "994. View: Adjust vertical zoom (MIDI CC/OSC only)"
- Since ReaLearn 2 and REAPER 6.20, there's special support for this type of actions. Starting from the first time this action is triggered, ReaLearn will track its current value.
- That's why toggling is supported. Because ReaLearn itself takes care of toggling, you need to set *Invoke* to "Absolute" and Mode to "Toggle button".
- Feedback is also supported.
- Toggling/feedback for this type of actions comes with some inherent limitations that are related to the fact that a) REAPER itself doesn't necessarily use actions to invoke its own functions and b) MIDI CC/OSC actions don't have the concept of a "current value" (unlike e.g. toggle actions or FX parameters).
- The bottom line of these limitations is that toggling/feedback will only work if the action itself is used to trigger the change and if the action is an absolute action (not relative).
- Limitations in detail:

1. In most cases, feedback will not work when changing the value in REAPER directly (e.g. when adjusting vertical zoom directly via the REAPER user interface).
2. It will only work for actions that support some kind of absolute value range (usually the case for all non-relative MIDI CC/OSC actions).
3. When the action is invoked via ReaLearn, the feedback will only work if "Invoke" is "Trigger" or "Absolute". It won't work with "Relative".
4. When the action is invoked from ReaScript or other extensions, it will only work if the invocation was done via `KBD_OnMainActionEx()` and an absolute value change.
5. When the action is invoked via a native REAPER action mapping, it will only work if the invocation is done using absolute MIDI CC/OSC (not relative).

Target "Project: Invoke transport action"

Invokes a transport-related action.

Action menu

Specifies which transport action should be invoked.

Play/stop

Starts playing the containing project if the incoming absolute control value is greater than 0%, otherwise invokes stop.

Play/pause

Starts playing the containing project if the incoming absolute control value is greater than 0%, otherwise invokes pause.

Stop

Stops the containing project if the incoming absolute control value is greater than 0%. Useful for distinguishing feedback between *paused* and *stopped* state.

Pause

Pauses the containing project if the incoming absolute control value is greater than 0%. Useful for distinguishing feedback between *paused* and *stopped* state.

Record

Starts/enables recording for the current project if the incoming absolute control value is greater than 0%, otherwise disables recording.

Repeat

Enables repeat for the containing project if the incoming absolute control value is greater than 0%, otherwise disables it.

Target "Project: Browse tracks"

Steps through tracks. To be used with endless rotary encoders or Incremental button mode.

Scroll TCP checkbox

See Target "Track: Select/unselect".

Scroll mixer checkbox

See Target "Track: Select/unselect".

Scope menu

Decides which tracks are considered and how.

All tracks
: Considers all tracks even those which are hidden.

Only tracks visible in TCP
: Considers only those tracks which are visible in the track control panel.

Only tracks visible in TCP (allow 2 selections)
: Like "Only tracks visible in TCP" but makes it possible to have 2 selections. One for the MCP and one for the TCP. These selections can be moved independently. This can make sense if you have a bunch of tracks that you only show in the TCP and another separate bunch of tracks that you only show in the MCP.

Only tracks visible in MCP
: Considers only those tracks which are visible in the mixer control panel.

Only tracks visible in MCP (allow 2 selections)
: See above.

Target "Project: Seek"

Allows you to use faders, knobs, encoders or incremental buttons to seek within portions of your project … with feedback that indicates the current position!

Feedback menu

Determines how frequently ReaLearn captures feedback and sends it to your feedback output.

Beat
: Roughly every beat.

Fast
: As fast as possible, thereby giving the satisfying feeling of continuity.

Behavior menu

Determines whether to use immediate or smooth seeking.

Seek play checkbox

Doesn't just change the edit cursor but also changes the play position when the project is currently being played.

Move view checkbox

Allow to scroll / change viewport when seeking.

"Use" checkboxes

The following checkboxes determine which time ranges will be taken into consideration as reference for seeking (control) and feedback.

If you don't tick any "Use" checkbox, ReaLearn will seek within the currently visible viewport.

If you tick multiple options, this is the order of fallbacks:

- If there's no time selection, the loop points will be used.
- If there are no loop points, the current region is used.
- If there's no current region, the project will be used.
- If the project is empty, the viewport will be used.

Use time selection checkbox

Can use the currently set time selection as reference.

Use loop points checkbox

Can use the currently set loop points as reference.

Use regions checkbox

Can use the current region as reference.

Use project checkbox

Can use the complete project as reference, from start to end.

Target-specific properties

This target supports the following additional target properties.

Name	Type	Description
`target.position.project_default`	String	Position in the current transport time unit
`target.position.time`	String	*minute:second.milli*
`target.position.measures_beats_time`	String	*measure.beat.milli*

Name	Type	Description
`target.position.measures_beats`	String	*measure.beat.milli*
`target.position.seconds`	String	*second.milli*
`target.position.samples`	String	*sample*
`target.position.hmsf`	String	*hour:minute:second:milli*
`target.position.absolute_frames`	String	*frames*
`target.position.project_default.mcu`	String	Like `target.position.project_default` but tailored to Mackie Control timecode displays
`target.position.time.mcu`	String	Like `target.position.time` but tailored to Mackie Control timecode displays
`target.position.measures_beats_time.mcu`	String	Like `target.position.measures_beats_time` but tailored to Mackie Control timecode displays
`target.position.measures_beats.mcu`	String	Like `target.position.measures_beats` but tailored to Mackie Control timecode displays
`target.position.seconds.mcu`	String	Like `target.position.seconds` but tailored to Mackie Control timecode displays
`target.position.samples.mcu`	String	Like `target.position.samples` but tailored to Mackie Control timecode displays
`target.position.hmsf.mcu`	String	Like `target.position.hmsf` but tailored to Mackie Control timecode displays
`target.position.absolute_frames.mcu`	String	Like `target.position.absolute_frames` but tailored to Mackie Control timecode displays

Target "Project: Set playrate"

Sets REAPER's master playrate.

This target doesn't currently work if the project containing ReaLearn is not the active project tab.

Target "Project: Set tempo"

Sets REAPER's master tempo.

This target is not learnable anymore via the "Learn target" button and also not eligible for the Target "Global: Last touched" because it causes too many "false positives".

Marker/region targets

Target "Marker/region: Go to"

Navigates to a specific marker or region. Here's the behavior in detail:

Regions

- If the project is stopped, the editor cursor immediately jumps to the start position of the given region.
- If the project is playing, playback will continue with the given region as soon as the currently playing region (or measure if not within a region) has finished playing. This is called "smooth seek".
- **Attention:** This currently doesn't work if the project containing ReaLearn is not the active project tab.

Markers

- If the project is stopped, the editor cursor immediately jumps to the given marker.
- If the project is playing, playback will immediately be continued at the given marker.

The advantage over REAPER's built-in actions is that this target allows to target arbitrarily many markers/regions (either by position or by ID) … and that it supports visual feedback! If you assign this target to a button which has an LED, you will see which marker/region is currently playing just by looking at your controller.

Please note that this doesn't work when recording!

Marker/region selector menu

This dropdown lets you choose if you want to refer to a marker/region by its user-assigned ID or by its position on the timeline.

Marker/region menu

This dropdown displays the markers or regions (depending on the *Regions* checkbox state).

Now! button

This sets the target to the currently playing (or currently focused, if stopped) marker/region.

Behavior menu

Determines whether to use immediate or smooth seeking.

Regions checkbox

Switches between markers and regions.

Set loop points checkbox

For regions, this will additionally set the loop points to the region start and end position.

Set time selection checkbox

For regions, this will additionally set the time selection to the region start and end position.

Target-specific properties

This target supports the following additional target properties.

Name	Type	Description
`target.bookmark.id`	Intger	(Numeric) ID of the bookmark
`target.bookmark.index`	Integer	Index of the bookmark (counting both markers and regions)
`target.bookmark.index_within_type`	Integer	Index of the bookmark (counting only markers or regions, respectively)
`target.bookmark.name`	String	Name of the bookmark
`target.bookmark.color`	Color	Custom color of the resolved marker or region.

Track targets

Target "Track"

A target that allows you to define a track.

Act/Tags controls

Act/Tags stands for "Action / Unit tags" and decides what happens when a control messages arrives, e.g. a button press.

Action menu

None (feedback only)

With this setting, nothing will happen. It's suited very well as neutral target for textual feedback with an expression that contains a track property, e.g. `{{ target.track.name }}`.

Set (as unit track)

The button press will set the track defined in this target as Unit track *without resolving it before*. For example, if this target defines to use the currently selected track (<Selected> selector), pressing the button will make the unit track dynamically reflect whatever track is selected.

Pin (as unit track)

The button press will resolve the track defined in this target and set the result as Unit track. For example, if this target defines to use the currently selected track, pressing the button will check

which track is currently selected and set the unit track to exactly this track. It will stay that way even if the user selects another track.

Unit tags field

The text field lets you define unit tags to determine for which units the Unit track should be changed. If it's empty, the current unit will be affected.

Target "Track: Arm/disarm"

Arms the track for recording if the incoming absolute control value is greater than 0%, otherwise disarms the track. This disables "Automatic record-arm when track selected". If you don't want that, use Target "Track: Select/unselect" instead.

Target "Track: Enable/disable all FX"

Enables all the track's FX instances if the incoming absolute control value is greater than 0%, otherwise disables them.

Target "Track: Enable/disable parent send"

Enables the parent send routing of the track if the incoming absolute control value is greater than 0%, otherwise disables it.

Target "Track: Mute/unmute"

Mutes the track if the incoming absolute control value is greater than 0%, otherwise unmutes the track.

Target "Track: Peak"

This is a feedback-only target! It turns your feedback-capable controller into a VU meter by constantly reporting the current volume of the configured track to it.

In addition to connecting it with a LED ring or motor fader source (which should be obvious), it can also be used with a single LED to build a clipping indicator:

1. Set *Target Min* to the minimum dB value that should make your clipping LED turn on. Leave *Target Max* at 12.00 dB.
2. Make sure the Out-of-range behavior menu is set to "Min or max".
3. If you have an LED that supports multiple colors, you will probably see a rainbow of colors flashing up which can be quite confusing. Use the feedback transformation formula `x = ceil(y)` to restrict the feedback to just two values: Min (0%) or Max (100%). You can then use Source Min/Max controls to adjust the off/on LED colors.

At the moment this target only reports peak volume, not RMS.

Target "Track: Phase invert/normal"

Inverts the track phase if the incoming absolute control value is greater than 0%, otherwise switches the track phase back to normal.

Target "Track: Select/unselect"

Selects the track if the incoming absolute control value is greater than 0%, otherwise unselects the track.

This target stops being learnable if you activate the REAPER preference "Mouse click on volume/pan faders and track buttons changes track selection" (because this preference would generate too many false positives). If you change the preference, ReaLearn will take it into consideration the next time you restart REAPER.

Scroll TCP checkbox

Also scrolls the track control panel to the desired track.

Scroll mixer checkbox

Also scrolls the mixer control panel to the desired track.

Target "Track: Set automation mode"

Sets the track to a specific automation mode if the incoming control value is greater than 0%, otherwise sets it back to REAPER's default track automation mode "Trim/Read".

Mode menu

Here you can pick the desired automation mode.

Target "Track: Set monitoring mode"

Sets the track to a specific input monitoring mode if the incoming control value is greater than 0%, otherwise sets it back to "Off".

Mode menu

Here you can pick the desired monitoring mode.

Target "Track: Set automation touch state"

When you use REAPER's "Touch" automation mode, REAPER needs a way to know if you are currently touching the control element which is bound to the automation envelope or not. As long as you keep touching it, it will overwrite existing automation. As soon as you release it, REAPER will leave the envelope untouched.

Classical control surfaces implement this very intuitively by providing touch-sensitive faders. With this target, you can easily reproduce exactly this behavior via ReaLearn. You do this by mapping the

touch event (which is usually nothing else than a MIDI note on/off message) to this target. The touch state is scoped to a particular track and parameter type which you can choose in the **Type** dropdown.

However, ReaLearn wouldn't be ReaLearn if it wouldn't allow you to let totally different sources take control of the touch state. For example, if you have a push encoder, you could map the "push" event to the touch state, allowing you to write automation only while you are touching the encoder. Or if you don't have a push encoder, you could just use some spare button.

Target "Track: Set pan"

Sets the track's pan value.

Target-specific properties

This target supports the following additional target properties.

Name	Type	Description
`target.pan.mcu`	String	Pan value tailored to one line on a Mackie Control LCD

Target "Track: Set stereo pan width"

Sets the track's width value (applicable if the track is in stereo pan mode).

Target-specific properties

This target supports the following additional target properties.

Name	Type	Description
`target.width.mcu`	String	Width value tailored to one line on a Mackie Control LCD

Target "Track: Set volume"

Sets the track's volume.

Target "Track: Show/hide"

Shows the track if the incoming absolute control value is greater than 0%, otherwise hides it.

Area menu

Lets you decide if you want it to show/hide in the track control panel or the mixer.

Target "Track: Solo/unsolo"

Soloes the track if the incoming absolute control value is greater than 0%, otherwise unsoloes the track.

Behavior menu

See the REAPER user guide for details.

Solo in place

Soloes the track while respecting REAPER's routing. This is REAPER's default and since ReaLearn v2.4.0 also ReaLearn's default.

Solo (ignore routing)

Soloes the track muting everything else, no matter the routing.

Use REAPER preference

Follows whatever is set in the REAPER preferences.

Learning this target by pressing the "Solo" button of the *master* track is currently not possible but of course you can just select it manually in the dropdown menu.

FX chain targets

Target "FX chain: Browse FXs"

Steps through the FX instances in the FX chain by always having exactly one FX instance visible. To be used with endless rotary encoders or previous/next-style "Incremental buttons".

Display menu

Here you can decide if you want to display the FX as part of the FX chain or in a dedicated floating window.

FX targets

Target "FX"

A target that allows you to define an FX, in its basic variant perfect for acquiring feedback for a specific FX.

Act/Tags controls

The setting **Act/Tags** allows you to optionally set/pin the declared FX as Unit FX. This works pretty much the same as described in Target "Track".

Target "FX: Enable/disable"

Enables the FX instance if the incoming absolute control value is greater than 0%, otherwise disables it.

Target "FX: Set online/offline"

Sets the FX instance online if the incoming absolute control value is greater than 0%, otherwise sets it offline.

Target "FX: Load snapshot"

Restores a certain state of a particular FX. Before using this target, you need to take a snapshot of the desired FX state using the **[Take!]** button. This snapshot will be saved as part of ReaLearn's state itself and as a direct consequence as a part of your project. This makes your project nicely self-contained. It's perfect for activating particular FX presets because it will always restore the desired state, even if the preset list has changed.

This target supports feedback, but only if the snapshot is loaded via ReaLearn itself.

Please note that some plug-ins have *very large* states. Therefore, you should keep an eye on the snapshot size, which will be displayed once you take the snapshot. ReaLearn's own state will grow with every new snapshot mapping, so this can quickly add up and make REAPER/ReaLearn slow!

Target "FX: Browse presets"

Steps through FX presets.

This target is suited for use with knobs, encoders and Incremental button mode because it allows you to step through the complete preset list. The minimum value always represents *No preset* whereas the maximum value always represents the last available preset.

It's *not* suited for activating a particular preset (e.g. by setting Target Min/Max controls to the same value), because the preset list of an FX is usually not constant. As soon as you modify the preset list, this value will might suddenly point to a completely different preset. Even worse, the actual preset might have been deleted.

If you want to activate a particular preset, please use the Target "FX: Load snapshot" instead.

Target "FX: Open/close"

Makes the FX instance visible if the incoming control value is greater than 0%, otherwise hides it.

Display menu

Here you can decide if you want to display the FX as part of the FX chain or in a dedicated floating window.

FX parameter targets

Target "FX parameter: Set automation touch state"

This does the same as Target "Track: Set automation touch state" but for FX parameter value changes.

Target "FX parameter: Set value"

Sets the value of a particular track FX parameter.

Parameter controls

The parameter to be controlled.

Please note that both Particular selector and At position selector address the FX by its position in the FX chain. The difference between the two is that Particular selector shows a dropdown containing the available parameters and At position selector lets you enter the position as a number in a text field. Latter is useful if at the time of choosing the position, the FX is not available.

Target-specific properties

This target supports the following additional target properties.

Name	Type	Description
`target.fx_parameter.index`	Integer	Zero-based index of the resolved FX parameter.
`target.fx_parameter.name`	String	Name of the resolved FX parameter.
`target.fx_parameter.macro.name`	String	Name of the corresponding Pot macro parameter. Only works if this parameter is part of a preset loaded via Pot.
`target.fx_parameter.macro.section.name`	String	Name of the corresponding Pot macro parameter section. Only works if this parameter is part of a preset loaded via Pot.
`target.fx_parameter.macro.section.index`	Integer	Zero-based index of the corresponding Pot macro parameter section (within the current bank). Only works if this parameter is part of a preset loaded via Pot.
`target.fx_parameter.macro.new_section.name`	String	Name of the corresponding Pot macro parameter section, but only if this parameter marks the start of a new section. Only works if this parameter is part of a preset loaded via Pot.
`target.fx_parameter.macro.bank.name`	String	Name of the corresponding Pot macro parameter bank. Only works if this parameter is part of a preset loaded via Pot.

Pot targets

Target "Pot: Browse filter items"

This target can be used to filter the potentially very large collection of presets in Target "Pot: Browse presets". The idea is to map this target to an endless rotary encoder or previous/next buttons (using Incremental button mode mode) and then navigate within the available filter items, e.g. instruments or banks.

Kind menu

Choose the kind of filter items that you want to browse. They correspond to the filters available in Pot Browser.

Target-specific properties

This target supports the following additional target properties.

Name	Type	Description
`target.item.name`	String	Name of the filter item.
`target.item.parent.name`	String	Name of the parent filter item if there's any. E.g. the instrument to which a bank belongs or the type to which a subtype belongs.

Target "Pot: Browse presets"

Use this target to browse a collection of presets. By default, this is the complete collection of presets available in all supported databases, so potentially thousands of presets. If you want to browse just a subset, see Target "Pot: Browse filter items".

The idea is to map this target to an endless rotary encoder or previous/next buttons (using Incremental button mode mode) and then navigate within the available presets. Once you have selected a preset, you can audition it via Target "Pot: Preview preset" (if it's a sound preset) and load it via Target "Pot: Load preset".

Target-specific properties

This target supports the following additional target properties.

Name	Type	Description
`target.preset.name`	String	Name of the preset.
`target.preset.product.name`	String	Name of the product to which this preset belongs, if available.
`target.preset.file_ext`	String	File extension of the preset, in case it's a file-based preset.
`target.preset.author`	String	Name of the preset author, if available.
`target.preset.vendor`	String	Name of the preset vendor, if available.
`target.preset.comment`	String	Preset comment, if available.

Target "Pot: Preview preset"

Auditions a preset selected via Target "Pot: Browse presets". Only works if it's a sound preset and a sound preview file is available.

Target "Pot: Load preset"

Loads a preset selected via Target "Pot: Browse presets".

This needs at least REAPER version 6.69+dev1030! Also, it only works if you have the VST2/VST2i version of the corresponding plug-in installed. - **NKS audio file presets:** Loading supported via ReaSamplOmatic5000

Track/FX controls

You must tell the target at which FX slot to load the corresponding plug-in. The best idea is to use FX selector At position selector. Selectors such as Particular selector or Named selector are not suited because the target might replace the plug-in with another one, in which the unique FX ID and the FX name can change. Then the target would turn inactive and stop working.

Target-specific properties

This target supports the same additional target properties as Target "Pot: Browse presets". The only difference is that the ones in this one relate to the currently loaded preset, not the one that's selected in the preset browser.

Send/receive targets

Target "Send: Automation mode"

Sets the track send to a specific automation mode if the incoming control value is greater than 0%, otherwise sets it back to REAPER's default automation mode "Trim/Read".

Target "Send: Mono/stereo"

Sets the track send to mono or back to stereo.

Target "Send: Mute/unmute"

Mutes/unmutes the track send.

Target "Send: Phase invert/normal"

Inverts the track send phase or switches it back to normal.

Target "Send: Set automation touch state"

This does the same as Target "Track: Set automation touch state" but for send volume or pan adjustments.

Target "Send: Set pan"

Sets the track send's pan value.

Target "Send: Set volume"

Sets the track send's volume.

Playtime targets

Target "Playtime: Slot management action"

TODO

Target "Playtime: Slot transport action"

TODO

Target "Playtime: Slot seek"

TODO

Target "Playtime: Slot volume"

TODO

Target "Playtime: Column action"

TODO

Target "Playtime: Row action"

TODO

Target "Playtime: Matrix action"

TODO

Target "Playtime: Control unit scroll"

TODO

Target "Playtime: Browse cells"

TODO

MIDI targets

Target "MIDI: Send message"

Sends arbitrary MIDI messages (also sys-ex!) in response to incoming messages. This target turns ReaLearn into a capable and convenient MIDI/OSC/Keyboard-to-MIDI converter.

Output menu

Where to send the MIDI message.

FX output

Sends the MIDI message to the output of this ReaLearn instance - which usually means it flows into the FX below ReaLearn, e.g. a VST instrument.

Feedback output

Sends the MIDI message to the device which is set as *output*.

Input device

Injects the MIDI message into the current MIDI input device buffer. Enables a unique feature called "Global MIDI transformation", as shown in tutorial video 11.

Device menu

When choosing output Input device, you can choose into which MIDI input device buffer the message will be injected.

<Same as input device>

Injects the message into the same buffer of the MIDI input device chosen as Input menu.

Specific input device

Injects the message into another specific MIDI input device. This can be useful for doing global MIDI transformation with controllers that expose multiple MIDI input ports. A practical example is shown in tutorial video 11.

Pattern field

Defines the MIDI message to be sent as Raw MIDI pattern. It allows you to encode the incoming *absolute* control value as part of the message (after it has been processed by Glue).

Pre-defined patterns menu (…)

Provides predefined patterns.

This is a target capable of real-time control!

This target is a bit special in that it carries out its processing logic exclusively in the audio thread if it's controlled by a MIDI source. This has the big advantage that receiving and producing MIDI messages happens in one go (without inter-thread-communication latency), which is often important when using MIDI message conversion.

However, this also means that the following things won't work when controlling this target using MIDI:

- It can't take the lead in Group interaction.
- It won't work with timed fire modes.

- If *output* is set to MIDI: <FX output>, additional limitations apply:
  - It can't act as a follower in Group interaction, either.
  - It can't participate in Target "ReaLearn: Load mapping snapshot".

OSC targets

Target "OSC: Send message"

Sends OSC messages with up to one argument in response to incoming messages. This target turns ReaLearn into a capable and convenient MIDI → OSC and OSC → OSC converter. If an argument number is entered (e.g. 1), it will encode the incoming absolute control value as that argument (after it has been processed by the glue section).

Output menu

Where to send the OSC message.

<Feedback output>

Sends the OSC message to the device which is set as *Output*. Of course this only works if it's an OSC device.

Specific device:

Sends the OSC message to a specific device.

Address, Argument and Range

These correspond to the identically named settings of Source "OSC". Check that section for details.

ReaLearn targets

Target "ReaLearn: Enable/disable instances"

This target allows you to flexibly enable or disable other ReaLearn instances based on the unit tags of their units.

Exclusivity menu

Non-exclusive

If the incoming control value is greater than 0%, all matching ReaLearn instances will be enabled (on top of the already enabled instances). If the value is 0%, all matching ReaLearn instances will be disabled.

Exclusive

If the incoming control value is greater than 0%, all matching ReaLearn instances will be enabled and all non-matching ones will be disabled. If the value is 0%, it's exactly the opposite (react to button press only if you don't want this to happen).

Exclusive (on only)

Variation of *Exclusive* that applies exclusivity only if the incoming control value is greater than 0%.

Tags field

A ReaLearn instance matches when at least one of its units is tagged with any of the unit tags entered into this field (comma-separated).

Remarks

- This affects other ReaLearn units only. It doesn't match against this unit.
- ReaLearn instances which don't contain units with tags won't be affected at all.
- Only affects instances in the same project. If *this* ReaLearn instance is on the monitoring FX chain, it only affects other instances in the monitoring FX chain.

This target is great for switching between completely different controller setups!

You enter unit tags here, but it will enable/disable whole instances! I know, this is counter-intuitive. In some cases, it would be good to have a way to enable/disable units. However, that doesn't exist yet. Create a feature request if you need that.

Target "ReaLearn: Dummy"

This target simply does nothing when invoked and also doesn't provide any meaningful feedback on its own.

It's sometimes useful to have such a dummy target, e.g. combined with Group interaction. Or if you want to use ReaLearn as a MIDI filter which just "eats" an incoming MIDI message. Or if you want to send some text feedback to a hardware display, if the text is just a constant string or uses a placeholder that doesn't need target context.

Target "ReaLearn: Enable/disable mappings"

This target allows you to flexibly enable or disable other mappings in this unit based on their tags:

Exclusivity menu

Non-exclusive

If the incoming control value is greater than 0%, all matching mappings will be enabled (on top of the already enabled mappings). If the value is 0%, all matching mappings will be disabled.

Exclusive

If the incoming control value is greater than 0%, all matching mappings will be enabled and all non-matching ones will be disabled. If the value is 0%, it's exactly the opposite (react to button press only if you don't want this to happen).

Exclusive (on only)

Variation of *Exclusive* that applies exclusivity only if the incoming control value is greater than 0%.

Tags field

A mapping matches when it is tagged with any of the mapping tags entered into this field (comma-separated).

Remarks

- This affects other mappings only, not *this* mapping.
- Mappings without tags won't be affected at all.

This target is a straightforward alternative to Conditional activation when it comes to bank switching!

Target "ReaLearn: Load mapping snapshot"

Restores target values for all or certain mappings in this ReaLearn unit.

Snapshot menu

Choose the snapshot that you want to load.

<Initial>

Restores the initial target values for the mappings.

By ID

Restores target values contained in a snapshot that was taken via Target "ReaLearn: Take mapping snapshot". Enter the corresponding ID here.

Default field

Allows you to define a default target value to restore for each participating mapping whenever the snapshot either doesn't exist or doesn't contain a value for that mapping. If that participating mapping has reverse checked, the inverse of the default value will be loaded.

Tags field

Allows you to restrict the set of mappings whose target values will be restored.

- If this field is empty, target values of all mappings will be restored.
- If this field contains tags (comma-separated), target values will be restored only for mappings that are tagged with any of these.

Active mappings only checkbox

By default, even target values for inactive (but control-enabled) mappings are restored! If you don't

like that, tick this checkbox.

Remarks

- Mappings for which control is not enabled, never participate in snapshotting.
- Some targets don't report values and therefore don't participate in snapshotting.
- Feedback of this target indicates whether the desired snapshot is the one which has last been loaded (for the given tags).

Target "ReaLearn: Modify mapping"

Triggers a modification of another ReaLearn mapping.

Kind menu

The kind of modification.

Learn target
: Switches "Learn target" on or off for the destination mapping. Use button **[…]** to pick the considered target types and invocations to be included in the learning process.

Set target to last touched
: Sets the target of the destination mapping to the last-touched target. Use button **[…]** to pick the considered target types and invocations.

Unit menu

Allows you to pick another ReaLearn unit.

Mapping menu

Allows you to pick the destination mapping.

This target is great to "pin" targets to certain control elements on demand.

Target "ReaLearn: Take mapping snapshot"

Memorizes target values for all or certain mappings in this ReaLearn units and saves them in a snapshot of your choice.

Snapshot menu

Choose the snapshot to which you want to save the mapping values.

<Last loaded>
: Always chooses the snapshot which is currently active (was last loaded) for the given tags.

  Only works if tags are not empty and if all tags have the same last-loaded snapshot. So the best is if you always enter exactly one tag.

By ID

Enter the unique ID of the snapshot, e.g. `scene_1`.

Tags field

Allows you to restrict the set of mappings whose target values will be memorized.

- If this field is empty, target values of all mappings will be memorized.
- If this field contains tags (comma-separated), target values will be memorized only for mappings that are tagged with any of these.

Active mappings only checkbox

By default, even target values of inactive (but control-enabled) mappings end up in the snapshot! If you don't like that, tick this checkbox.

Target "ReaLearn: Browse group mappings"

This target lets you choose an arbitrary mapping group in this compartment and cycle through it with an encoder/fader/knob or incremental (previous/next) buttons.

"Cycling through" means that you move from one mapping in the group to the next one by hitting the next mapping's target with the *Target Max* value in its glue section (by default 100%).

Group menu

The group that you want to browse.

Exclusivity menu

Non-exclusive

Really just hits the target of the mapping which is next in the line and doesn't do anything with the other mappings. In many cases this is enough, e.g. if the targets of the mappings in the cycled group are the same and just "Target Max" is different. Or if the target itself already takes care of exclusivity.

Exclusive

Doesn't just hit the target of the mapping which is next in the line but also hits the targets of all other mappings in the cycled group with their respective *Target Min* value (by default 0%). Be careful with this, you often won't need it.

Inactive mappings are skipped!

A mapping group lends itself perfectly for defining things that should happen *in sequence*. This target allows you to take advantage of that!

- Combine it with Target "ReaLearn: Enable/disable mappings" to browse different banks.
- Combine it with Target "ReaLearn: Enable/disable instances" to browse

completely different controller setups (or banks).

- Combine it with targets that don't provide a "Browse ..." variant themselves.
- Use it as an alternative to target value sequences that allows you to have completely different targets within one sequence.

Target "Virtual"

This is exactly the counterpart of the possible virtual sources in the Main compartment. Choosing a Virtual target here is like placing cables between a Real control element and all corresponding main mappings that use this Virtual control element as source.

Learnable checkbox

If you disable this checkbox, this virtual source will not be learnable via Learn source in the main compartment. This can be useful for rather unimportant control element interactions such as *Fader touch* that would otherwise make it very hard to learn more important sources such as *Fader movement*.

Further concepts

This section describes further concepts. You may or may not need to understand them, it depends on which ReaLearn features you are going to use and how complex your control scenarios are.

General concepts

Import/export

ReaLearn can import and export data in 2 formats:

JSON

A wide-spread data exchange format. It's a text format, so if you are familiar with the search & replace feature of your favorite text editor, this is one way to do batch editing.

Luau

A full-blown programming language derived from the famous Lua language that is also used in REAPER itself.

For the programmers and script junkies out there: It's perfectly possible to program ReaLearn from outside by passing it a snippet of JSON via `TrackFX_SetNamedConfigParm()`. Parameter name is `set-state`. This mechanism is implemented on ReaLearn side using REAPER's named parameter mechanism (search for `named_parameter_name`).

Example that assumes that the first FX of the first track is a ReaLearn instance:

```
local track = reaper.GetTrack(0, 0)
local state = [[
{
  "controlDeviceId": "62",
  "feedbackDeviceId": "fx-output",
  "mappings": [
    {
      "name": "1",
      "source": {
        "type": 1,
        "channel": 0,
        "number": 64
      },
      "mode": {},
      "target": {
        "type": 2
      }
    }
  ]
}
]]
```

```
reaper.TrackFX_SetNamedConfigParm(track, 0, "set-state", state)
```

Feedback relay

Feedback *relay* happens when a feedback-enabled mapping becomes inactive and another feedback-enabled mapping with the same Source becomes active. In that case, ReaLearn has to swap the displayed value of the previous mapping target with the displayed value of the new mapping target.

Instance concepts

Instance ID

A randomly assigned ID that uniquely identifies a particular Instance. Will most likely change after a restart of REAPER!

Auto units

Each Instance optionally supports the automatic addition and configuration of a Unit with a user-defined main preset if a certain type of controller is connected and automatic removal if it is disconnected.

The general procedure is:

1. Globally define once what controllers you have at your disposal and choose which main preset you want to use for which controller (in the Helgobox App)
2. Enable global control for one Helgobox instance using Enable global control.

It's a good idea to enable global control for a Helgobox instance on the monitoring FX chain. Such an instance will be around permanently as long as REAPER is running, even if you open and close different projects. Perfect for project-spanning control scenarios!

And now the nice part: If you decide to use a specific device for something else in a certain project, all you need to do is to set use the device as input and/or output in a project-specific ReaLearn unit! If you do that, the project-specific instance "wins" over the monitoring FX instance. You got a project-specific override. If you close the project, the monitoring FX instance takes over again.

Playtime

Playtime is a modern session view / clip launcher for REAPER, built straight into Helgobox.

Each Helgobox Instance may contain one *Playtime Matrix* (by default not loaded)

Pot Browser

Pot Browser is an experimental modern preset browser built straight into Helgobox. It's just a prototype so far. It will probably look quite different in the future. You can open it via menu action Open Pot Browser.

It’s recommended to use Pot Browser from a ReaLearn instance on the monitoring FX chain, that way you have the browser accessible from any project.

Add a toolbar button which triggers the REAPER action "ReaLearn: Open first Pot Browser" to get quick and convenient access to the browser.

Remarks:

- Pot Browser is in an experimental stage, it doesn’t save any of your settings!
- Each ReaLearn instance can have one *Pot Unit* (by default not loaded). Each Pot Unit has its own filter and preset state. When you open the Pot Browser from an instance, it connects to the Pot Unit of that instance.
- ReaLearn’s "Pot" targets such as Target "Pot: Browse presets" can be used to control the Pot Unit from any controller.

Unit concepts

Letting through events

ReaLearn by default "eats" incoming MIDI events for which there’s at least one active mapping with that source. In other words, it doesn’t forward MIDI events which are used to control a target parameter. However, unmatched MIDI events are forwarded! You can change this using Let through section.

The exact behavior differs depending on what you choose as Input menu:

- If input is set to MIDI: <FX input>
  - MIDI events arrive from ReaLearn’s FX input. If they get forwarded, they get forwarded to the FX output, usually to the plug-in which is located right below ReaLearn FX. The default setting often makes much sense here, especially if you put ReaLearn right above another instrument plug-in.
- If input is set to a MIDI hardware device
  - MIDI events arrive directly from the MIDI hardware device. If they get forwarded, they get forwarded to REAPER’s tracks as they would usually do without ReaLearn. If they don’t get forwarded, it means they get filtered and will never make it to the tracks. ReaLearn completely eats them, globally! That means, ReaLearn can act as global MIDI filter.
  - Please note, with input set to a real MIDI device, MIDI events coming from *FX input* are *always* forwarded to the FX output.
  - Also, MIDI events captured from a real MIDI device input are **never** forwarded to ReaLearn’s FX output.

    This global MIDI filter feature is only available in REAPER v6.36+.
- If input is set to an OSC device
  - You won’t see the checkboxes because they don’t make sense for OSC.

- The checkboxes don't have any effect on computer keyboard input. Keys are always passed through when doing text entry and never passed through if a mapping matches.

Auto-load

If you activate Auto-load based on unit FX, ReaLearn will start to observe the Unit FX of this ReaLearn unit and keep loading main presets according to which FX-to-preset links you have defined. By default, the unit FX is set to `<Focused>`, which means, it will reflect whatever FX is currently focused. Whenever the unit FX changes, it will check if you have linked a compartment preset to it and will automatically load it. Whenever the unit FX switches to an unlinked FX or the FX loses focus, ReaLearn falls back to the mapping list or preset that was active before activating auto-load.

Of course this makes sense only if you actually have linked some presets. Section Unit FX-to-preset link describes how to do that.

FX-to-preset link

A link between a FX and a Main preset. Used in Auto-load.

Unit FX-to-preset link

A link saved as part of a Unit.

Global FX-to-preset link

This is like a Unit FX-to-preset link but the link is saved globally. This is useful if you have only one controller or if you have x controllers (= and therefore x ReaLearn units) and want both of them to always auto-load the same preset if the unit FX points to the same plug-in.

- All links will be saved *globally*, not just within this project!
- Location: REAPER resource directory (**Options › Show REAPER resource path in explorer/finder**) at `Data/helgoboss/realearn/auto-load-configs/fx.json`.

Unit key

Each ReaLearn unit has a key that's used to address this particular ReaLearn unit when using the Projection feature. By default, the unit key is a random cryptic string which ensures that every unit is uniquely addressable. The result is that scanning the QR code of this ReaLearn unit will let your mobile device connect for sure with this unique unit, not with another one - remember, you can use many units of ReaLearn in parallel. This is usually what you want.

But a side effect is that with every new ReaLearn unit that you create, you first have to point your mobile device to it in order to see its Projection (by scanning the QR code). Let's assume you have in many of your projects exactly one ReaLearn unit that lets your favorite MIDI controller control track volumes. By customizing the unit key, you can tell your mobile device that it should always show the Projection of this very ReaLearn unit - no matter in which REAPER project you are and even if they control the volumes of totally different tracks.

You can achieve this by setting the unit key of each volume-controlling ReaLearn unit to exactly the same value, in each project, using Unit data… button. Ideally it's a descriptive name without spaces, such as "track-volumes". You have to do the pairing only once et voilà, you have a dedicated device for monitoring your volume control ReaLearn units in each project.

Make sure to not have more than one ReaLearn unit with the same unit key active at the same time because then it's not clear to which your mobile device will connect!

At the moment, the unit key is part of the ReaLearn preset! That means, opening a preset, copying/cutting a ReaLearn FX, importing from clipboard - all of that will overwrite the unit key. This might change in future in favor of a more nuanced approach!

Unit track

The second line of the bottom panel shows the current track chosen as **Unit track** for this unit of ReaLearn. This can be something like "Track 3" or "The currently selected track". Mappings in this ReaLearn unit can refer to this track by choosing the track selector Unit selector.

The unit track can be changed via Target "Track".

Unit FX

The second line of the bottom panel also shows the current FX chosen as **Unit FX** for this unit of ReaLearn. This can be something like "FX 5 on track 3" or "The currently focused track". Mappings in this ReaLearn unit can refer to this FX by choosing the FX selector Unit selector.

The unit FX can be changed via Target "FX".

Unit tag

Each unit can have arbitrarily many tags.

Tags are important if you want to dynamically enable or disable instances using the Target "ReaLearn: Enable/disable instances".

Projection

Projection is a quite unique feature that allows you to project a schematic representation of your currently active controller to a mobile device (e.g. a tablet computer). You can put this device close to your controller in order to see immediately which control element is mapped to which parameter. This is an attempt to solve an inherent problem with generic controllers: That it's easy to forget which control element is mapped to which target parameter.

Logging

Logging can be enabled or disabled via Logging Menu.

Logging of real control messages

Each log entry contains the following information:

- Timestamp in seconds
- Helgobox Instance ID
- Message purpose
  - **Real control:** A message used for controlling targets.
  - **Real learn:** A message used for learning a source.
- Actual message (MIDI messages will be shown as hexadecimal byte sequence, short MIDI messages also as decimal byte sequence and decoded)
- Match result
  - **unmatched:** The message didn't match any mappings.
  - **matched:** The message matched at least one of the mappings.
  - **consumed:** Only for short MIDI messages. This short message is part of a (N)RPN or 14-bit CC message and there's at least one active mapping that has a (N)RPN or 14-bit CC source. That means it will not be processed. The complete (N)RPN or 14-bit CC message will be.

Logging of real feedback messages

The log entries look similar to the ones described above, with the following notable differences.

- Message purpose
  - **Feedback output:** A message sent to your controller as response to target value changes.
  - **Lifecycle output:** A message sent to your controller as response to mapping activation/deactivation (see Mapping lifecycle actions).
  - **Target output:** A message sent because of either the Target "MIDI: Send message" or Target "OSC: Send message".

Superior units

When a unit is made superior via menu entry Make unit superior, this unit is allowed to suspend other units which share the same input and/or output device (hardware devices only, not FX input or output!).

Making units superior is **rarely needed**!

This option was initially introduced in order to add more flexibility to the Auto-load feature. The idea was to let a controller *fall back* to some default behavior if the currently focused FX is closed. Multiple instances were necessary to make this work with one of them (the auto-load instance) being superior.

However, since ReaLearn 2.14.0, falling back to initial mappings when the FX loses focus in auto-load mode became much easier and doesn't require multiple units anymore! Your initial mappings or initial preset will be memorized and reloaded

> once the FX loses focus. See Auto-load for more information.

Behavior:

- By default, ReaLearn units are not superior, just normal. This is most of the time okay, even if you have multiple units that share the same input and output … as long as you don't have any conflicting mappings active at the same time.
- For example, if 2 units use the same input or output device, and they use different control elements, they can peacefully coexist. And even if they share a control element for the *control direction*, they are still fine with it. The same control element will control 2 mappings, why not!
- Things start to get hairy as soon as 2 units want to send *feedback* to the same control elements at the same time. You should avoid this. You should not even do this within one ReaLearn unit. This can't work.
- Sometimes you want one unit to suspend/cover/cancel/mute another one! You can do this by making this unit *superior*. Then, whenever this unit has at least one active mapping, all non-superior units with the same control and/or feedback device will be disabled for control and/or feedback.
- You can have multiple superior units. Make sure they get along with each other :)

Compartment concepts

Compartment preset

Main presets vs. controller presets

Main preset

The term *main preset* is just a shortcut for saying "compartment preset in the main compartment".

Controller preset

The term *controller preset* is just a shortcut for saying "compartment preset in the controller compartment".

Factory preset vs. User preset

Factory preset

Factory presets are built-in compartment presets. You can't change them yourself. But you can "make them your own" by making a copy of them. See Writing presets with Luau.

User preset

User presets are made by users, for example by you.

- Saving your mappings as a preset is optional. All controller mappings are saved together with your current ReaLearn unit anyway, no worries. But as soon as you want to reuse these mappings in other ReaLearn unit or for Auto-load, it makes of course sense to save them as a

preset!

- All of your presets end up in the REAPER resource directory (REAPER → Options → Show REAPER resource path in explorer/finder) at `Data/helgoboss/realearn/presets` followed by `main` (for main compartment presets) or `controller` (for controller compartment presets). They are JSON files and very similar to what you get when you press *Export to clipboard*.
- They can even be in a subdirectory. Please note that the subdirectory name becomes a part of the preset ID, so better don't move existing presets around if you want preset references of existing ReaLearn units to stay intact.
- JSON files can also contain custom data sections. For example, the ReaLearn Companion app adds a custom data section to controller presets in order to memorize the positions and shapes of all control elements.

Writing presets with Luau

It is possible to write compartment presets with the Luau language instead of building them via the user interface. Many of the more complex ReaLearn factory presets are written in Lua, e.g. the "DAW control" preset.

A good way to get started writing Luau presets is to create your personal compartment preset user workspace.

A preset workspace is a subdirectory within the compartment preset parent directory that may contain a bunch of presets and other files.

Important facts about preset workspaces/namespaces:

- It may contain both Luau presets (`.preset.luau`) and conventional JSON presets (`.json`)!
- The name of the workspace (subdirectory) is at the same time the first part of the preset ID. For example, if the subdirectory name is `helgoboss` and it contains a preset file `my-preset.json`, the final ID of that preset will be `helgoboss/my-preset`.
- That also means that presets from different workspaces never conflict with each other.
- Therefore, a preset "workspace" is at the same time a preset "namespace". Those terms are sometimes used interchangeably.
- It's important that the ID of a preset doesn't change, especially if you want to use that preset with Auto-load. If you change the ID, it's another preset from ReaLearn's perspective!
- Conversely, the name of the workspace directory and the name/path of the preset file within the workspace directory should not change!
- The only thing that is allowed to change is the file extension. This makes it possible to convert a preset from JSON to Luau and vice versa.
- Preset workspaces are self-contained. What does that mean? Luau presets can use the `require` statement to share common Luau code. However, this is only possible within one preset workspace.
- As a result, it is safe to have multiple completely different preset workspace, and it's guaranteed that they don't conflict with each other. This makes preset sharing easy (it's just a matter of copying the preset workspace directory).

- There's one special preset workspace: The *user workspace*. It's the workspace whose directory has the same name as your macOS/Windows/Linux user. Special features:
  - The user workspace is where ReaLearn puts your presets when you save them via the user interface (as `.json` files).
  - All `require` statements in Luau code imported via **Import from clipboard** are resolved against this user workspace.

You can create a preset workspace by pressing **Menu → Compartment presets → Create compartment preset workspace (including factory presets)** (done for each compartment type separately). This will create a randomly-named preset workspace directory within the compartment preset parent directory. If this is your first preset workspace, it is best practice to turn it into your personal *user workspace* by renaming the generated directory to your macOS/Windows/Linux username (name must match exactly!).

Maybe the user workspace directory exists already. Most likely because you have saved presets from the user interface, in which case it should contain only JSON files. In that case you can safely move all files and directories from the generated preset workspace directory into that existing directory.

The generated workspace contains:

- A README file with some general information and tips.
- A copy of all ReaLearn factory presets for that compartment.
  - Mainly Luau presets (ending with `.preset.luau`).
  - You can use them as inspiration for your own ones.
  - Most of the factory presets in the main compartment are quite advanced. One of the easier ones is `generic/numbered/fx-paraameters.preset.luau`.
- A bunch of Luau SDK files in the first directory level of the workspace.
  - They contain Luau types and utility functions.
  - You can require them within your own Luau files in that workspace and use them to build presets.
  - However, the usage of the SDK files is completely optional! The only important thing about building ReaLearn presets is that the returned table conforms to the ReaLearn compartment API (= has the structure that you get when you do **Export from clipboard → Export … compartment as Lua**). It doesn't matter if you use Luau's type system to build that table or the provided utility functions or your own or none.
  - The SDK files can change in incompatible ways in newer ReaLearn versions. Only ReaLearn's built-in compartment API is guaranteed to stay backward-compatible!

Luau presets have a YAML frontmatter comment section right at the top of the file that contain meta information about the preset. The following properties are possible:

`name`	required	Preset display name

`realearn_version`	required	The ReaLearn version for which this preset was built. This can effect the way the preset is loaded, e.g. it can lead to different interpretation or migration of properties. So care should be taken to set this correctly!
`author`		Preset author
`description`		Preset description. Preferably in Markdown format, but can also be plain text.
`setup_instructions`		Setup instructions. Preferably in Markdown format, but can also be plain text.
`device_manufacturer`	controller compartment only	Manufacturer of the device represented by the controller preset.
`device_name`	controller compartment only	Name of the device represented by the controller preset.
`midi_identity_pattern`	controller compartment only	MIDI identity compatibility pattern. Will be used for auto-adding controllers and for finding the correct controller preset when calculating auto-units.
`midi_output_port_patterns`	controller compartment only	Possible MIDI identity compatibility patterns. Will be used for auto-adding controllers and for finding the correct controller preset when calculating auto-units. It should only be provided if the device in question doesn't reply to device queries or if it exposes multiple ports which all respond with the same device identity and only one of the ports is the correct one. Example: APC Key 25 mk2, which exposes a "Control" and a "Keys" port. ReaLearn will match any in the list. OS-prefixes are alowed, e.g. `macos:` will only match on macOS.

`provided_schemes`	controller compartment only	Provided virtual control schemes. Will be used for finding the correct controller preset when calculating auto units. The order matters! It directly influences the choice of the best-suited main presets. In particular, schemes that are more specific to this particular controller (e.g. "novation/launchpad-mk3") should come first. Generic schemes (e.g. "grid") should come last. When auto-picking a main preset, matches of more specific schemes will be favored over less specific ones.
`used_schemes`	main compartment only	Used virtual control schemes. Will be used for finding the correct controller preset when calculating auto units.
`required_features`	main compartment only	A set of features that a Helgobox instance needs to provide for the preset to make sense. Will be used for determining whether an auto unit should be created for a specific instance or not. Example: If the required feature is "playtime" and a controller is configured with this main preset but the instance doesn't contain a Playtime Clip Matrix, this instance will not load the main preset. Currently, only feature `playtime` is supported, which matches if the Helgobox instance contains a Playtime Matrix.

Compartment parameter

Each ReaLearn compartment contains 100 freely assignable parameters. Compartment parameters can be used in the following ways:

- For Conditional activation (lets the parameter value influence which mappings are active)
- For Dynamic selector (lets the parameter value influence which object (track, FX etc.) is targeted for a specific mapping)
- As Source "ReaLearn parameter" (lets the parameter value control any ReaLearn target)

They can be customized as described in Compartment parameters menu. Parameter customizations are saved together with the compartment preset. Parameter values will be reset whenever you load a preset (just the ones in that compartment).

Continuous vs. discrete compartment parameters

By default, compartment parameters have a Continuous value range. Although that makes them very versatile, it's often easier to work with a Discrete value range.

Entering a value count (see Value count) makes the parameter have a Discrete value range with the given number of integer values. For example, a value count of 10 means that the parameter can represent exactly 10 values (0 to 9).

Choose the value count wisely and think twice before changing it to a different value at a later point in time!

You probably want to refer to values of this parameter in certain parts of ReaLearn, e.g. in Target Min/Max controls. If you do that and later change the value count, these value references will not be valid anymore. They will point to other integers than you intended to. So if you are not sure, better pick a large value count and stick to it!

Compartment-wide Lua code

Each compartment may contain arbitrary Luau code to be reused by multiple mapping MIDI source and feedback scripts. This avoids code duplication and decreases memory usage. It even allows the usage of shared state.

The code that you provide here is treated as a module that MIDI source and feedback scripts can import using `require("compartment")`. That means you need to export everything that you want the MIDI source and feedback scripts to see, simply by returning it.

Example 5. Compartment-wide Lua code

The following compartment-wide Lua code exports 2 functions named `get_text` and `get_number`:

```
local module = {}

local function private_function()
    return "i'm private"
end

function module.get_text()
    return "hello world"
end

function module.get_number()
    return 5
end

return module
```

These functions can then be reused in MIDI source and feedback scripts:

```
local compartment = require("compartment")
local text = compartment.get_text()
local number = compartment.get_number()
```

Compartment-wide Lua code is part of the compartment, that means it's also saved as part of a compartment preset!

Virtual control

Virtual control make it possible to create main presets that can be reused with many different controllers.

The idea is simple:

1. You define a Controller preset for a DAW controller, mapping each Real control element (e.g. its first fader, which emits MIDI CC7 messages) to a corresponding Virtual control element (e.g. named `ch1/fader`) by using a Virtual target.
2. You define a Main preset, mapping each Virtual control element to some Real target by using a Virtual source. For example, you map `ch1/fader` to the Target "Track: Set volume".
3. ReaLearn creates a sort of wire between the Controller compartment and the Main compartment. So you can now control the track volume by moving the first fader.
4. Most importantly, the main preset is now generic because it's not built for a specific controller anymore!

See Using the controller compartment for more information how to do this in detail!

Virtual feedback

Virtual feedback is just like Virtual control, but in the opposite direction (from REAPER to your Controller).

Real vs. virtual control elements

Wait … control elements that are not real!? Yes! In ReaLearn, they exist. We can distinguish between Real control element and Virtual control element.

Real control element

A *real* control element is an element that really exists on a Controller, e.g. a fader that you can touch.

Virtual control element

A *virtual* control element is an abstraction of a Real control element. It enables Virtual control and Virtual feedback.

Each virtual control element has a *type* and an *ID*.

Virtual control element ID

A number or name that uniquely identifies the control element on the device.

Numbers are especially suited for the 8-knobs/8-buttons layouts. In a row of 8 knobs one would typically assign number 1 to the leftmost and number 8 to the rightmost one. It's your choice.

For more advanced virtual control scenarios it can be useful to think in names instead of numbers. You can use up to 32 alphanumeric and punctuation characters (no exotic characters, e.g. no umlauts).

Virtual control element type

If you want to define a virtual control element, you should first decide which type is should have: *Multi* or *Button*. This distinction is used by ReaLearn to optimize its user interface.

For numbered control elements, the type is even part of the Virtual control element ID. For example, "Multi 1" is considered a different virtual control element than "Button 1". For named control elements, this is not the case. `col1/row1/pad` defined as Multi is considered the same as `col1/row1/pad` defined as Button.

Multi

Represents a control element that you can "move", that is, something that allows you to choose between more than 2 values. Usually everything which is *not* a simple on/off button :) Here's a list of typical *multis*:

- Fader
- Knob
- Pitch wheel
- Mod wheel
- Endless encoder
- XY pad (1 axis)
- Touch strip
- Rotary (endless) encoder
- Velocity-sensitive pads or keys

Button

Represents a control element that distinguishes between two possible states only (e.g. on/off), or even just one ("trigger"). Usually it has the form factor of a button that you can "press". Here's a list of typical *buttons*:

- Play button
- Switch
- Sustain pedal (a simple on/off one, not a half-pedaling one!)

Please note that velocity-sensitive keys should be exposed as Button - unless you know for sure that you are not interested in the velocity sensitivity.

Virtual control scheme

If you want your main preset to be compatible with as many controller presets as possible, try to use predefined names instead of inventing your own virtual control naming scheme.

When you define a virtual source or virtual target, there's a convenient picker that provides names for the following standardized virtual control schemes:

DAW control (`daw`)

The names you see here are heavily inspired by the wording used on Mackie Control devices.

Grid (`grid`)

For controls divided into rows and column, as for example found on the Novation Launchpad.

Numbered (`numbered`)

Simply lets you pick among any number between 1 and 100.

Order in which mappings are processed

Since ReaLearn 2.10.0, mappings are processed from top to button, exactly in the order in which they are defined within the corresponding compartment. This matters if you want to map multiple targets to one button and the order of execution matters.

Important: There's an exception. ReaLearn's processing of its own VST parameters is always deferred.

- That means changing a ReaLearn parameter in one mapping and relying on it in the next one (in terms of conditional activation or in a `<Dynamic>` expression), will not work!
- You can work around that by delaying execution of the next mapping via fire mode but that's a dirty hack. ReaLearn's parameters are not supposed to be used that way!
- Imagine a railway: ReaLearn's targets can be considered as trains. Triggering a target means moving the train forward. ReaLearn's parameters can be considered as railway switches. Changing a parameter means setting a course. The course needs to be set in advance, at least one step before! Not at the same time as moving the train over the switch.

Mapping concepts

Absolute vs. relative control

We can distinguish between two fundamentally different ways of Control: *Absolute* and *relative* control.

Absolute control

Imagine someone tells you: "Change the volume to -6 dB!" You go ahead and move the fader to -6

dB.

That's absolute control in a nutshell! Absolute control uses absolute control values.

Often, absolute control involves an Absolute control element, but it doesn't have to!

ReaLearn can simulate absolute control even with a Relative control element (see Make absolute).

Relative control

Imagine someone tells you: "Raise the volume by 2 dB!" You go ahead and move the fader up by 2 dB. Before it was -6 dB, that means it's -4 dB now.

That's relative control! Relative control uses relative control values.

Often, relative control involves a Relative control element, but it doesn't have to! ReaLearn masters multiple ways of turning absolute control into relative one:

- Buttons or keys naturally emit absolute control values. But using Incremental button mode, they can be used for relative control (previous/next style buttons).
- Knobs naturally emit absolute control values. But using Make relative or Performance control, they can be used for relative control.

Absolute vs. relative control elements

We can distinguish between control elements with regard to what sort of Control value they emit: In this dimension, we have Absolute control element and Relative control element.

Absolute control element

A Control element is *absolute* if it emits absolute values. You can think of an absolute value as a percentage: The value is something between 0% and 100%, where 0% represents the minimum possible value and 100% the maximum.

Typical absolute control elements

Fader

A fader is a vertical or horizontal element with a thumb that you can move. When moving a fader up from bottom to top, it will continuously emit values from 0% to 100%.

Knob

A knob is a circular element with boundaries on the left and right. When moving a knob clockwise, it will continuously emit values from 0% (at the left boundary) to 100% (at the right boundary).

Momentary button

A momentary button is a button that emits 100% when pressed and 0% when released. ReaLearn can easily use momentary buttons to toggle a target. See Toggle button mode.

Toggle button

A toggle button is a button that emits 100% when pressed and 0% when pressed again.

You don't want toggle buttons on the hardware side! They are much less flexible than momentary buttons! Momentary buttons can be turned into toggle buttons by ReaLearn, but not vice versa! If you have to put up with a hardware toggle button, use MIDI source character Toggle-only button.

Velocity-sensitive key

A key emits a value between 0% and 100% when pressed, depending on the velocity with which the key was hit. When released, it will emit 0%.

Aftertouch

When pressing the key a bit more into the keybed after it has already been pressed, it will continuously emit increasing values starting at 0%. When releasing pressure, it will continuously emit decreasing values until reaching 0% again.

Pitch wheel

When moving a pitch wheel up, it will continuously emit increasing values starting at 50%. When letting it snap back, it will continuously emit decreasing value until reaching 50% again. When moving it down, it will continuously emit decreasing values, and so on.

Touch strips

When touching the strip somewhere in the middle, it will emit a value around 50%. When dragging upward, it will continuously emit increasing values starting from where you touched it first.

Relative control element

A Control element is *relative* if it emits relative values. You can think of a relative value as an *instruction*. It can be one of the following two instructions:

Please decrease!

We call such a value a **decrement**.

Please increase!

We call such a value an **increment**.

Typical relative control elements

Rotary endless encoder

A rotary endless encoder (or just *encoder*) is a circular, like a knob. But unlike a knob, it doesn't have boundaries. When moving a rotary endless encoder clockwise, it will continuously emit increments. When moving it counter-clockwise, it will continuously emit decrements.

It happens very often that controllers have rotary endless encoders, but they will act like knobs by default, sending absolute messages. That is a great waste, and you should change that setting as soon as possible on the hardware side.

Control value

A *control value* is the signal that travels through a ReaLearn Mapping from Source to Target when it receives an event from an Input port. A control value can be absolute or relative.

Feedback value

A *feedback value* is the signal that travels through a ReaLearn Mapping back from Target to Source when the target changes its value. A feedback value is always absolute.

Absolute vs. relative control values

Absolute control value

An *absolute* control value is conceptually a percentage between 0.0% and 100.0%.

Internally, it is represented by a high-precision floating point number between 0.0 and 1.0. E.g. 0.25 is 25%.

Relative control value

A *relative* control value is a number of increments or decrements.

Internally, it is represented as a positive or negative integer. E.g. control value -2 means a decrement of 2.

Mapping tag

Each mapping can have arbitrarily many tags. Such tags can be used to organize mappings in a way that is much more flexible than groups.

Tags are not just something for people that love to keep things tidy! They also get meaning in combination with certain ReaLearn targets such as Target "ReaLearn: Enable/disable mappings".

Mapping group

Mapping groups are part of the currently shown compartment and enable you to divide the list of mappings into multiple groups.

Groups can be useful …

- To apply an activation condition to multiple mappings at once.
- To enable/disable control/feedback for multiple mappings at once.
- To keep track of mappings if there are many of them.

You can decide which group is displays using Group menu.

You can move existing mappings between groups by opening the context menu (accessible via right-click on Windows and Linux, control-click on macOS) of the corresponding mapping row and choosing "Move to group".

Groups are saved as part of the project, VST plug-in preset and compartment preset.

Mapping activation state

A mapping is considered as **on** or **active** (terms are used interchangeably) only if all following criteria are fulfilled:

1. The mapping is complete, that is, both source and target are completely specified
2. The mapping is enabled as a whole
3. The mapping has control and/or feedback enabled
4. The Mapping activation condition is fulfilled
5. The Target activation condition is fulfilled
6. The target is valid

> Example: A track target can be invalid when it's using <Selected> selector but no track is currently selected).

In all other cases, mapping is **off** or **inactive**. In that case, it doesn't have any effect!

(Controller) mappings with Virtual target are always considered active as long as the feedback checkbox is ticked.

Mapping signal flow

Here's how ReaLearn processes an incoming control event that matches a mapping source.

1. ReaLearn converts the event coming from the Input port to a Control value.
2. ReaLearn feeds the Control value to the mapping's Glue. The glue section is responsible for transforming control values before they reach the Target. This transformation can change the type of the control value, e.g. from relative to absolute - it depends on the settings in the glue section and the mapping's target. The glue section can even "eat" control values so that they don't arrive at the target at all.
3. Finally, ReaLearn converts the transformed Control value into some target instruction (e.g. "set volume to -6.0 dB") and executes it.

Feedback (from target to source) works in a similar fashion but is restricted to absolute control values. Even if the source is relative (e.g. an encoder), ReaLearn will always emit absolute feedback, because relative feedback doesn't make sense.

Conditional activation

Conditional activation is a powerful feature that allows you to dynamically activate or deactivate a mapping depending on its Mapping activation condition.

Example 6. Control A when a button is not pressed, control B when it is

Here's how you would implement a typical use case. You want your rotary encoder to control target A when the button is not pressed and control target B when it's pressed.

1. Create a mapping for the button
    - As "Target", you need to choose ReaLearn itself (Type: Target "FX parameter: Set value", Track: `<This>`, FX: "… VSTi: ReaLearn (Helgoboss)"). As "Parameter", choose an arbitrary ReaLearn parameter, e.g. "Parameter 1".
    - As "Mode", choose either "Absolute" (if you want to switch the encoder function just momentarily) or "Toggle" (if you want the button to toggle between the two encoder functions).
2. Create a mapping with target A
    - Set "Active" to "When modifiers on/off", "Modifier A" to "Parameter 1" and disable the checkbox beside it. Set "Modifier B" to `<None>`.
    - This basically means "Hey, ReaLearn! Please activate this mapping only if ReaLearn Parameter 1 is **off**!" (remember, we control ReaLearn Parameter 1 using the button).
    - At this point, turning your encoder should control target A, but only if you don't press the button!
3. Create a mapping with target B
    - Just as in step 2, set "Active" to "When modifiers on/off" and "Modifier A" to "Parameter 1". **But**: Now **enable** the checkbox beside it. Set "Modifier B" to `<None>`.
    - This basically means "Hey, ReaLearn! Please activate this mapping only if ReaLearn Parameter 1 is **on**!"
    - At this point, turning your encoder should control target A if you don't press the button and control target B if you press the button.

Mapping activation condition

The activation condition of a mapping determines under which circumstances a mapping is active or inactive, based on the value of a Compartment parameter or based on the state of arbitrary targets. It is especially practical if your controller has a limited amount of control elements and you want to give control elements several responsibilities. It lets you easily implement use cases such as:

- "This knob should control the track pan, but only when my sustain pedal is pressed, otherwise it should control track volume!" (modifier use cases)
- "I want to have two buttons for switching between different banks where each bank represents a group of mappings." (bank use cases)

- "I want to control the volume of this track only if it's not muted." (target-state based use cases)

Since ReaLearn 2.11.0, Target "ReaLearn: Enable/disable mappings" provides a slightly less powerful but more straightforward way to implement use cases that were before only achievable with parameter-based conditional activation.

There are 6 different activation modes:

- **Always:** Mapping is always active (the default)
- **When modifiers on/off:** Mapping becomes active only if something is pressed / not pressed
- **When bank selected:** Allows you to step through different groups of mappings (sometimes also called "pages")
- **When EEL met** Let an EEL formula decide (total freedom)
- **When expression met:** Let an expression decide (total freedom)
- **When target value met:** Let the current value of the target of another mapping decide

At this occasion, some words about ReaLearn's own freely assignable FX parameters.

ReaLearn itself isn't just able to control parameters of other FX, it also offers FX parameters itself. At the moment it offers 200 FX parameters, 100 for the main compartment and 100 for the controller compartment. You can control them just like parameters in other FX:

- Via automation envelopes,
- via track controls,
- via REAPER's own MIDI/OSC learn
- … and of course via ReaLearn itself.

Initially, they don't do anything at all. First, you need to give meaning to them by referring to them in activation conditions or `<Dynamic>` selector expressions.

When modifiers on/off

This mode is comparable to modifier keys on a computer keyboard. For example, when you press `Ctrl+V` for pasting text, `Ctrl` is a modifier because it modifies the meaning of the `V` key. When this modifier is "on" (= pressed), it activates the "paste text" and deactivates the "write the letter V" functionality of the `V` key.

In ReaLearn, the modifier is one of the FX parameters. It's considered to be "on" if the parameter has a value greater than 0 and "off" if the value is 0.

You can choose up to 2 modifier parameters, "Modifier A" and "Modifier B". If you select "<None>", the modifier gets disabled (it won't have any effect on activation). The checkbox to the right of the dropdown lets you decide if the modifier must be "on" for the mapping to become active or "off".

Example: The following setting means that this mapping becomes active *only* if both "Parameter 1" and "Parameter 2" are "on".

- **Modifier A:** "Parameter 1"
- **Checkbox A:** Checked
- **Modifier B:** "Parameter 2"
- **Checkbox B:** Checked

Now you just have to map 2 controller buttons to "Parameter 1" and "Parameter 2" via ReaLearn (by creating 2 additional mappings - in the same ReaLearn instance or another one, up to you) et voilà, it works. The beauty of this solution lies in how you can compose different ReaLearn features to obtain exactly the result you want. For example, the *absolute mode* of the mapping that controls the modifier parameter decides if the modifier button is momentary (has to be pressed all the time) or toggled (switches between on and off everytime you press it). You can also be more adventurous and let the modifier on/off state change over time, using REAPER's automation envelopes.

When bank selected

This is the correct activation mode if you want control surface "bank-style" mapping.

For this kind of use cases you should consider the new Target "ReaLearn: Enable/disable mappings", which is available since ReaLearn 2.11.0 as an alternative. It's slightly less powerful than conditional activation but probably easier to use, partly because you can dictate which mappings should be active "from outside", not from the perspective of the mapping itself.

You can tell ReaLearn to only activate your mapping if a certain parameter has a particular value. The particular value is called "Bank". Why? Let's assume you mapped 2 buttons "Previous" and "Next" to increase/decrease the value of the parameter (by using "Incremental button" mode, you will learn how to do that further below). And you have multiple mappings where each one uses "When bank selected" with the same parameter but a different "Bank". Then the result is that you can press "Previous" and "Next" and it will switch between different mappings within that parameter. If you assign the same "Bank" to multiple mappings, it's like putting those mappings into one group which can be activated/deactivated as a whole.

Switching between different programs via "Previous" and "Next" buttons is just one possibility. Here are some other ones:

- **Browse banks using a rotary encoder:** Just map the rotary encoder to the "Bank" parameter and restrict the target range as desired.
- **Activate each bank with a separate button:** Map each button to the "Bank" parameter (with absolute mode "Normal") and set "Target Min/Max" to a distinct value. E.g. set button 1 min/max both to 0% and button 2 min/max both to 1%. Then pressing button 1 will activate bank 0 and pressing button 2 will activate bank 1.

In previous versions of ReaLearn you could use other methods to achieve a similar behavior, but it always involved using multiple ReaLearn instances:

- **By enabling/disabling other ReaLearn instances:** You can use one main ReaLearn instance containing a bunch of mappings with Target "FX: Enable/disable" in order to enable/disable other ReaLearn FX instances. Then each of the other ReaLearn instances acts as one mapping bank/group.
- **By switching between presets of another ReaLearn instance:** You can use one main ReaLearn instance containing a mapping with Target "FX: Browse presets" in order to browse presets of another ReaLearn FX instance. Then each preset in the other ReaLearn instance acts as one mapping bank/group. However, that method is pretty limited and hard to maintain because presets are something global (not saved together with your REAPER project).

With *Conditional activation* you can do the same (and more) within just one ReaLearn unit.

If you want to adjust the number of banks and improve bank handling in general, set a discrete value count for the corresponding bank parameter (see Compartment parameter).

When EEL met

This is for experts. It allows you to write a formula in EEL2 language that determines if the mapping becomes active or not, based on potentially all parameter values. This is the most flexible of all parameter-based activation modes. The other modes can be easily simulated. The example modifier condition scenario mentioned above written as formula would be:

```
y = p1 > 0 && p2 > 0
```

`y` represents the result. If `y` is greater than zero, the mapping will become active, otherwise it will become inactive. `p1` to `p100` contain the current parameter values. Each of them has a value between 0.0 (= 0%) and 1.0 (= 100%).

This activation mode accounts for ReaLearn's philosophy to allow for great flexibility instead of just implementing one particular use case. If you feel limited by the other activation modes, just use EEL.

For most activation conditions which need this amount of freedom, the newer activation mode When expression met is a slightly better choice because it's easier to use and generally performs a bit better.

When expression met

This is very similar to the previous EEL activation mode. But instead of EEL, it lets you use the same expression language as used in dynamic selectors to express the activation condition.

The equivalent expression to above EEL example is:

```
p[0] > 0 && p[1] > 0
```

When target value met

This is different from all the other activation condition types in that it doesn't look at ReaLearn's internal parameter values. Instead, it looks at the target of another mapping (the so-called "lead mapping") and switches our mapping (the so-called "follow mapping") on or off depending on the target value of the lead mapping.

It works like this:

1. Create the lead mapping and give it a target, e.g. Target "Track: Select/unselect".
    - This lead mapping doesn't need to have a source. It can even be completely disabled!
2. In the **Mapping** dropdown, pick this newly created mapping.
3. In the **Expression** text field to the right, enter `y > 0`.
    - This means you want the follow mapping to be active whenever the target value of the lead mapping is greater than 0.0. Or in other words, when it's "switched on".

You can detect an inactive target by using `y == none`.

Raw MIDI pattern

Source "Raw MIDI / SysEx" and Target "MIDI: Send message" allow to enter so-called MIDI patterns, an advanced ReaLearn concept.

Pattern basics

In its most basic form, the pattern is a sequence of bytes notated as hexadecimal numbers. This is typical notation, especially for system-exclusive MIDI messages.

Example 7. A SysEx pattern

```
F0 00 20 6B 7F 42 02 00 10 77 00 F7
```

If you enter this and ReaLearn receives this system-exclusive message from the input, it will fire a 100% value. If feedback is set up correctly, this message will be sent to the device whenever the target value changes.

Remarks:

- You can check if the correct feedback messages are sent to the device by enabling Logging of real feedback messages.
- Each byte is written using 2 hexadecimal digits.
- Spaces between the bytes can be omitted.
- You can express all types of MIDI messages using this raw notation (e.g. pitch wheel), not just system-exclusive ones. If you do this, it will work as expected for the *feedback* direction. Please note that it will not work for the *control* direction at the moment (I don't think this is needed).

- If you want a system-exclusive MIDI message, you *must* include its start (`F0`) and end status byte (`F7`)!

Binary notation

ReaLearn also supports binary notation of a byte. You need to enclose the binary digits of one byte in brackets.

Example 8. Binary notation

```
F0 00 20 [0110 1011] 7F 42 02 00 10 77 00 F7
```

This is equivalent to the previous example (`6B` in hexadecimal notation is the same as `0110 1011` in binary notation).

Remarks:

- Between the brackets, each digit represents one bit. The left bit is the most significant one.
- Spaces between the two nibbles (4 bits) can be omitted.

Variable patterns (extracting and encoding a value)

For the Feedback direction, the examples I've shown you so far aren't real-world examples, because there's no point in sending the same MIDI message to the device over and over again! If you really would want to send a constant MIDI message to the device, you would be much better off using a Mapping lifecycle action, which allow you to send raw MIDI messages once when a mapping is initialized, not on every target value change.

But even for the Control direction, you might want to react to a whole *range* of system-exclusive messages, not just a fixed one. One part of your message might represent a variable value. You might want to extract it and control the target with it.

Fortunately, ReaLearn offers a uniform way to extract a variable value from the raw MIDI message (control) or encode the current target value into the raw MIDI message (feedback). Bytes which contain a variable value (or a part of it) *must* be expressed using binary notation.

Example 9. Variable pattern

```
F0 00 20 6B 7F 42 02 00 10 77 [0000 dcba] F7
```

The second nibble of the second last byte contains the lowercase letters `dcba`. This is the portion of the byte that denotes the variable value.

Each letter represents one bit of the variable value:

`a` Bit 1 (least significant bit of the variable value)

`b` Bit 2

`c` Bit 3

`d` Bit 4 …

`m` Bit 13

`n` Bit 14

`o` Bit 15

`p` Bit 16 (most significant bit of the variable value)

Resolution of variable patterns

The resolution of the variable value always corresponds to the letter in the whole pattern which represents the highest bit number. In the example above, the resolution is 4 bit because there's no letter greater than `d` in the pattern.

Example 10. Another variable pattern

In this example, the resolution is 7 bit because `n` is the greatest letter in the whole pattern.

```
F0 00 20 6B 7F 42 02 00 10 [00nm lkji] [hgfe dcba] F7
```

Remarks:

- The highest resolution currently supported is 16 bit (= 65536 different values).
- You can put these letter bits anywhere in the pattern (but only within bytes that use binary notation).

Byte order

This form of notation is slightly unconventional, but I think it's very flexible because it gives you much control over the resulting MIDI message. This amount of control seems appropriate considering the many different ways hardware manufacturers used and still use to encode their MIDI data.

When a number is expressed within more than one byte, manufacturers sometimes put the most significant byte first and sometimes the least significant one, there's no rule. This notation supports both because you decide where the bits end up:

Example 11. Most significant byte first

```
F0 00 20 6B 7F 42 02 00 10 [ponm lkji] [hgfe dcba] F7
```

Example 12. Least significant byte first

```
F0 00 20 6B 7F 42 02 00 10 [hgfe dcba] [ponm lkji] F7
```

More examples

Example 13. "Romeo and Juliet" bits (separated by 2 bytes)

```
F0 [1111 000b] [a101 0100] F7
```

Example 14. Simple on/off value (1 bit only)

```
F0 A0 [1111 010a] F7
```

Example 15. Pitch wheel simulation

This behaves like pitch wheel because the pattern describes exactly the way how pitch wheel messages are encoded.

```
E0 [0gfe dcba] [0nml kjih]
```

Source concepts

Real vs. virtual sources

We distinguish between *virtual* and *real* sources.

Virtual source

A *virtual* source refers to a Virtual control element and can only be used in the Main compartment.

Examples: `ch1/fader`

Real source

A *real* source refers to a Real control element.

MIDI source character

MIDI control-change messages serve a very wide spectrum of MIDI control use cases. Even though some control-change controller numbers have a special purpose according to the MIDI specification

(e.g. CC 7 = channel volume), nothing prevents one from using them for totally different purposes. In practice that happens quite often, especially when using general-purpose controllers. Also, there's no strict standard whatsoever that specifies how relative values (increments/decrements) shall be emitted and which controller numbers emit them.

Therefore, you explicitly need to tell ReaLearn about it by setting the *source character*.

The good news is: If you use "Learn source", ReaLearn will try to guess the source character for you by looking at the emitted values. Naturally, the result is not always correct. The best guessing result can be achieved by turning the knob or encoder quickly and "passionately" into clockwise direction. Please note that guessing doesn't support encoder type 3.

The possible source characters are:

Range element (knob, fader, etc.)

A control element that emits continuous absolute values. Examples: Fader, knob, modulation wheel, pitch bend, ribbon controller. Would also include a endless rotary encoder which is (maybe unknowingly) configured to transmit absolute values.

Button (momentary)

A control element that can be pressed and emits absolute values. It emits a > 0% value when pressing it and optionally a 0% value when releasing it. Examples: Damper pedal.

Encoder (relative type *x*)

A control element that emits relative values, usually an endless rotary encoder. The *x* specifies *how* the relative values are sent. This 1:1 corresponds to the relative modes in REAPER's built-in MIDI learn:

Type 1

- 127 = decrement; 0 = none; 1 = increment
- 127 > value > 63 results in higher decrements (64 possible decrement amounts)
- 1 < value <= 63 results in higher increments (63 possible increment amounts)

Type 2

- 63 = decrement; 64 = none; 65 = increment
- 63 > value >= 0 results in higher decrements (64 possible decrement amounts)
- 65 < value <= 127 results in higher increments (63 possible increment amounts)

Type 3

- 65 = decrement; 0 = none; 1 = increment
- 65 < value <= 127 results in higher decrements (63 possible decrement amounts)
- 1 < value <= 64 results in higher increments (64 possible increment amounts)

Toggle-only button (avoid!)

A control element that can be pressed and emits absolute values. ReaLearn will simply emit 100%, no matter what the hardware sends.

This is a workaround for controllers that don't have momentary buttons! You should only use

this character if there's absolutely no way to configure this control element as a momentary button.

Background

ReaLearn can make a momentary hardware button work like a full-blown toggle button. Its toggle mode is inherently more powerful than your controller's built-in toggle mode!).

However, the opposite is not true. It can't make a toggle hardware button act like a momentary button.

Combination with Incremental button mode

If you use the toggle-only source character in combination with mode Incremental button mode, you must leave source max at the (default) theoretical maximum value for that source (e.g. 127 for MIDI CC). Even if your controller device only sends 0 and 1 and in all other mappings you would enter the controller's concrete (instead of theoretically possible) maximum value. Otherwise, for this special case, a fixed out-of-range-behavior will set in that will just ignore all button presses.

MIDI source script

MIDI source scripts are EEL or Luau scripts to configure the Source "MIDI Script".

General mechanics

Each script receives an input and must produce an output.

Script input

- The main input is the current feedback value, which the script can access as a variable.

Script output

- The main output that the script is supposed to return is the MIDI message to be sent to the MIDI device.
- Additionally, the script can provide a so-called *feedback address*, which is supposed to uniquely identify the LED, motor fader or display.

  It's important to provide an address if you want ReaLearn to handle feedback relay correctly, e.g. that it switches off the LED when not in use anymore and doesn't switch it off if another mapping "takes over" the same LED. By convention, the constant (non-variable) bytes of the MIDI message should be used as address. The examples below might help to understand.

EEL script specifics

Scripts written in EEL work as follows.

Script input

- EEL scripts can access numeric feedback values only. The current numeric feedback value is

available as variable `y`, a floating point number between 0.0 and 1.0. This is essentially the current normalized target value after being processed by the "Glue" section of the mapping.

Script output

- In order to provide the output MIDI message, you must assign the raw bytes of that message to subsequent slots of the EEL script's virtual local address space (by indexing via brackets) **and** set the variable `msg_size` to the number of bytes to be sent. If you forget the latter step, nothing will be sent because that variable defaults to zero!
- In order to provide the address, simply assign an appropriate number to the `address` variable.

Example 16. Creating a 3-byte MIDI message

```
address = 0x4bb0;
msg_size = 3;
0[] = 0xb0;
1[] = 0x4b;
2[] = y * 64;
```

Luau script specifics

Scripts written in Luau work as follows.

Script input

- Luau scripts can access numeric, text and dynamic feedback values.
- Here's the list of input variables:

`y`
: The feedback value, either numeric (`type(y) == "number"`) or text (`type(y) == "string"`).

`context.feedback_event.color`
: The color as set in the Glue section section. Either the default color (`== nil`) or an RGB color (table with properties `r`, `g` and `b`).

`context.feedback_event.background_color`
: The background color.

Script output

- A Luau script can even generate multiple output messages.
- You need to return a table which contains the following keys:

`address`
: The feedback address.

`messages`
: An array containing all the messages, where each message itself is an array contaning the

message bytes.

Example 17. Creating a 3-byte MIDI message, assuming that `y` is a numeric normalized value.

```
return {
    address = 0x4bb0,
    messages = {
        { 0xb0, 0x4b, math.floor(y * 10) }
    }
}
```

Example 18. Creating a MIDI sys-ex message that changes the color of some fictional device LED/display.

```
local color = context.feedback_event.color
if color == nil then
    -- This means no specific color is set. Choose whatever you need.
    color = { r = 0, g = 0, b = 0 }
end
return {
    address = 0x4b,
    -- Whatever messages your device needs to set that color.
    messages = {
        { 0xf0, 0x02, 0x4b, color.r, color.g, color.b, 0xf7 }
    }
}
```

Example 19. Creating a 3-byte MIDI message, assuming that `y` is a text value.

```
local lookup_table = {
    playing = 5,
    stopped = 6,
    paused = 7,
}
return {
    messages = {
        { 0xb0, 0x4b, lookup_table[y] or 0 }
    }
}
```

Please note that this kind of simple mapping from text values to integer numbers doesn't need a script. You can use the `feedback_value_table` Glue section property instead, which can only be set via API though. Do a full-text search for `feedback_value_table` in directory `resources/controller-presets` of the ReaLearn source code to find usage examples.

You can share code between multiple MIDI scripts by using Compartment-wide Lua code, with the following limitations (which hopefully will be lifted over time):

- The shared code is not yet available to the Lua code editor window. That means writing `require("compartment")` will evaluate to `nil` in the editor. You might see a corresponding error message when the editor tries to compile your code.

OSC feedback arguments expression

This expression is used to enable for more flexible feedback for the Source "OSC".

It allows you to define exactly which feedback value is sent at which argument position. If this field is non-empty, the argument type will be ignored for the Feedback direction.

The format of this field is very simple: You enter feedback value property keys separated by spaces. Each entered property key corresponds to one argument position.

Example 20. Custom feedback with 2 arguments

If you want ReaLearn to send the current feedback value in text form at argument 1 and the color (see Feedback style menu (…)) as RRGGBB string at argument 2, you would enter:

```
value.string style.color.rrggbb
```

The following properties are available:

Property	Type	Description
`value.float`	`Float`	Numeric feedback value interpreted as float
`value.double`	`Double`	Numeric feedback value interpreted as double
`value.bool`	`Bool`	Numeric feedback interpreted as bool (on/off only)
`value.string`	`String`	Numeric or textual feedback value formatted as string
`style.color.rrggbb`	`String`	Feedback value color formatted as RRGGBB string
`style.background_color.rrggbb`	`String`	Feedback value background color formatted as RRGGBB string
`style.color`	`Color`	Feedback value color as native OSC color
`style.background_color`	`Color`	Feedback value background color as native OSC color

Property	Type	Description
`nil`	`Nil`	Nil value
`inf`	`Infinity`	Infinity value

Glue concepts

Target value sequence

A target value sequence represents a list of possible target values. It can be entered using into the Value sequence field.

The mapping will set the target only to values contained in that sequence. Such a sequence doesn't just support single values but also ranges with customizable step sizes. All values are entered comma-separated using the target unit specified with the Display unit button.

Example 21. Single values

Enter this sequence for a volume target with target unit switched to **dB**:

```
-20, -14, -12, -6, -3.5, 0
```

When you move your knob or rotary encoder or press a button using Incremental button mode, ReaLearn will step through the entered dB values for you.

Example 22. Value ranges

Enter this sequence for a target with a continuous value range and target unit switched to **%**:

```
10 - 30, 50 - 70 (5), 80 - 90 (2)
```

It will first step in 1% steps from 10% to 30%, then in 2% steps from 50% to 70% and finally from 80% to 90% in 2% steps. It's important that the numbers and the range dash are separated by spaces!

Example 23. Non-monotonic or non-strict-monotonic sequences

Let's look at this sequence:

```
20, 10, 10, -5, 8
```

It's non-monotonic: It decreases, and then increases again. Even if it would just decrease, it would be non-strict monotonic because it contains duplicates (value 10).

When using Absolute control, it's no problem stepping through such sequences.

However, Relative control only supports strictly increasing or strictly decreasing sequences. So if you control this sequence e.g. via Rotary endless encoder or via Incremental button mode, the sequence will be stepped through like this: -5, 8, 10, 20.

Alternative: Use Make absolute!

Feedback type

The *feedback type* determines whether to send numeric, text or dynamic feedback to the source. It can be set using the Feedback type controls.

Numeric feedback: EEL transformation

Sends numeric feedback to the source. This is the default.

Numeric feedback can be combined with an EEL feedback transformation formula. This is similar to Control transformation (EEL) field but used for translating a target value back to a source value for feedback purposes.

Be aware: Here `x` is the desired source value (= output value) and `y` is the current target value (= input value), so you must assign the desired source value to `x`.

Example 24. Simple feedback transformation formula

```
x = y * 2
```

ReaLearn's feedback processing order is:

1. Apply target interval.
2. Apply reverse.
3. Apply transformation.
4. Apply source interval.

Text feedback: Text expression

With this option, ReaLearn will send text feedback values to the source. This only works with sources that are capable of displaying text: That is any Source "OSC" with argument type *String*, Source "Display" and Source "MIDI Script".

Text feedback can be combined with a *text expression*, which lets you define which text is going to be sent to the source *whenever the target value changes* and immediately when entering the text. Whatever text you enter here, will be sent verbatim to the source.

Of course, entering a fixed text here is not very exciting. Most likely you want to display dynamic text such as the name of the currently selected track or the current target value, nicely formatted! You can do that by using placeholders, delimited by double braces.

Example 25. Simple text expression

```
{{target.text_value}}
```

See Target property for a list of properties that you can use in placeholders.

Dynamic feedback: Lua script

This feedback type puts you fully in charge about which feedback to send to the source. It does so by letting you define a Luau script that builds numeric, textual or even arbitrarily structured feedback.

General mechanics

ReaLearn executes your script whenever one of the ReaLearn-provided properties used in your script might have changed its value.

The script receives an input and must produce an output.

Script input

- The input is a function `context.prop` which you can use to query arbitrary properties, e.g. target or mapping properties. Those properties are the very same properties that you can use in textual feedback.

  Example 26. How to use `context.prop()`

  ```
  local preset_name = context.prop("target.preset.name")
  local param_name = context.prop("target.fx_parameter.name")
  ```

- Values returned by this function can be `nil`! E.g. target-related properties return a `nil` value whenever the mapping or target turns inactive, which is a very common situation. So it's important to prepare your Luau code for that, otherwise script execution fails and no feedback will be sent. One way to deal with a `nil` value returned by `context.prop` is to also return `nil` as `value` (see below).

Script output

- The output that the script is supposed to return is a table which looks as in the following example.

  Example 27. Result table structure

  ```
  return {
      feedback_event = {
          -- The feedback value ①
          value = "Arbitrary text",
          -- An optional color ②
          color = { r = 0, g = 255, b = 0 },
          -- An optional background color ③
          background_color = nil,
      }
  }
  ```

  ① In this example it's a text value, but it can be anything!

② Has the same effect as color in Feedback style menu (…)

③ Has the same effect as background color in Feedback style menu (…)

- The most important thing here is `value`. It can either be …
  - … a string (ideal for display sources)
  - … a number (ideal for LEDs and motor faders)
  - … `nil` (which means "turn the source off", e.g. turn off the LED, turn down the motorfader, clear the display text)
  - … or anything else (`true`, `false` or an arbitrary table … at the moment, this is only useful for the Source "MIDI Script" because other sources don't know how to deal with it)

Example 28. `global.realearn.time`

Displays the number of milliseconds passed since ReaLearn was loaded:

```
local millis = context.prop("global.realearn.time")
return {
    feedback_event = {
        value = "" .. millis .. "ms"
    },
}
```

Example 29. Animation

Creates an animation to make a long FX name visible on a tiny screen:

```
function create_left_right_animation(global_millis, max_char_count, frame_length,
text)
    if text == nil then
        return nil
    end
    if #text > max_char_count then
        local frame_count = #text - max_char_count
        local frame_index = math.floor(global_millis / frame_length) %
(frame_count * 2)
        local text_offset
        if frame_index < frame_count then
            text_offset = frame_index
        else
            local distance = frame_index - frame_count
            text_offset = frame_count - distance
        end
        return text:sub(text_offset + 1, text_offset + max_char_count)
    else
        return text
```

```
    end
end

-- The maximum number of characters we want to display at once
local max_char_count = 10
-- How many milliseconds to remain in one position
local frame_length = 150
local millis = context.prop("global.realearn.time")
local fx_name = context.prop("target.fx.name")
local animation = create_left_right_animation(millis, 10, frame_length, fx_name)
return {
    feedback_event = {
        value = animation
    },
}
```

Example 30. Structured feedback values

Returns a structured feedback value …

```
return {
    feedback_event = {
        value = {
            available = context.prop("target.available"),
            more_info = {
                index = context.prop("target.discrete_value"),
                count = context.prop("target.discrete_value_count"),
            },
        }
    },
}
```

… which can then be processed by a Source "MIDI Script":

```
return {
    address = 0x4bb0,
    messages = {
        { 0xb0, 0x4b, y.more_info.index, y.more_info.count }
    }
}
```

This example is not realistic, it just shows how you can access the value table returned by the glue section feedback script.

You can share code between multiple feedback scripts by using Compartment-wide Lua code, with the following limitations (which hopefully will be lifted over time):

- The shared code is not yet available to the Lua code editor window. That means writing `require("compartment")` will evaluate to `nil` in the editor. You might see a corresponding error message when the editor tries to compile your code.
- When ReaLearn queries the script in advance to know which target properties it needs, the shared code is also not available yet. Currently, you need to make sure that the target properties are queried even if `require("compartment")` evaluates to `nil`.

Target concepts

Real vs. virtual targets

We distinguish between *virtual* and *real* targets.

Virtual target

A *virtual* target controls a Virtual control element and can only be used in the Controller compartment.

Example: `ch1/fader`

It's then picked up by a Virtual source] in the Main compartment.

Real target

All others targets are real.

Examples: Target "Track: Set volume"

Target object selectors

Many ReaLearn targets work on some kind of *object*:

- A track
- An FX
- An FX parameter
- A send or receive

All of those objects need to be *addressed* somehow. For this purpose, ReaLearn uses so-called *object selectors*.

Common object selectors

This section describes commonly available object selectors.

The descriptions below are somewhat tailored to track objects. However, the same concepts can easily be applied to other objects that support these selectors.

Unit selector

This selector makes the target work on the current Unit track or current Unit FX of this particular ReaLearn Unit.

Particular selector

Lets you pick a specific object (e.g. track) and refer to it by its unique ID. This is the default. Choose this if you want ReaLearn to always control that very particular track even in case you move it somewhere else or rename it.

Please note that this is an extremely sticky selector. It's *not possible* with this setting to create a ReaLearn preset that is reusable among different projects. Because a track ID is globally unique, even across projects. That also means it doesn't make sense to use this setting in a ReaLearn Unit on the monitoring FX chain.

At position selector

This is the most straightforward selector. It lets you refer to a track by its position in the track list. This is great if you want to build a preset that you are going to reuse among multiple projects.

However, this selector has the disadvantage that things fall apart if you reorder, insert or delete tracks. This is why it's not the default.

Next to the dropdown you will find a text field. Here you should enter the position as number, starting with number `1`.

Named selector

Allows you to choose a track depending on its name. In case there are multiple tracks with the same name, it will always prefer the first one. This will allow you to use one ReaLearn preset across multiple projects that have similar naming schemes, e.g. as monitoring FX.

In the name field next to the dropdown, you can enter a name.

If you don't want exact matching, you can use wildcards:

`*` for matching zero or arbitrary many characters

`?` for matching exactly one arbitrary character.

Example 31. Wildcards in named selectors

`Violin *` would match `Violin 1` or `Violin 12` but not `12th Violin`.

Dynamic selector

This selector allows you to *calculate* which object (e.g. track) you want to use.

When you choose this option, a text field will appear next to it. This lets you enter a mathematical expression whose result should be the object's *index*. E.g. the first track in the project has index 0,

the master track -1. For your convenience, you will find a small text label next to the expression text field that always shows the current result of your formula (clamped to the target value range).

Please note that the expression language is *not* EEL - this is a notable difference to ReaLearn's control/feedback transformation text fields! The expression language used here just provides very basic mathematical operations like addition (`+/-`), multiplication (`*`) etc. and it also doesn't allow or need any assignment to an output variable.

The dynamic selector is a very powerful tool because you can use some special variables:

Variable	Type	Applicable to	Description
`none`	-	All selectors	Special value that represents a "not set" value. See below for examples.
`p`	Array of floating-point numbers	All selectors	Allows you to access the values of ReaLearn's internal parameters. E.g. you can get the number of the first parameter by writing `p[0]`. By default, parameter values are normalized floating point values, that means they are decimal numbers between 0.0 and 1.0. This can be changed by customizing the parameter with a specific integer value count (see Compartment parameter).
`p1 ⋯ p100`	Floating-point numbers	All selectors	Values of ReaLearn's internal parameters as single variables. *Deprecated*: Recent ReaLearn versions offer the `p` array instead. Better use that one.
`selected_track_index`	Integer >= -1	Track selectors	Resolves to the zero-based index of the first currently selected track within the containing project. If no track is selected, this resolves to `none`. If the master track is selected, `-1`.
`selected_track_tcp_index`	Integer >= -1	Track selectors	Like `selected_track_index` but counts only tracks that are visible in the track control panel.
`selected_track_mcp_index`	Integer >= -1	Track selectors	Like `selected_track_index` but counts only tracks that are visible in the mixer control panel.

Variable	Type	Applicable to	Description
`selected_track_indexes`	Array of integers >= -1	Track selectors	Lets you access the indexes of multiple selected tracks. E.g. if 2 tracks are selected, `selected_track_indexes[0]` gives you the index of the first selected track whereas `selected_track_indexes[1]` gives you the index of the second selected track. `selected_track_indexes[2]` would resolve to `none`.
`this_track_index`	Integer >= -1	Track selectors	Resolves to the zero-based index of the track on which this ReaLearn instance is located.
`instance_track_index`	Integer >= -1	Track selectors	Resolves to the zero-based index of the instance track of this ReaLearn instance.
`instance_track_tcp_index`	Integer >= -1	Track selectors	Like `instance_track_index` but counts only tracks that are visible in the track control panel.
`instance_track_mcp_index`	Integer >= -1	Track selectors	Like `instance_track_index` but counts only tracks that are visible in the mixer control panel.
`this_fx_index`	Integer >= 0	FX selectors	Resolves to the zero-based index of this ReaLearn FX instance.
`instance_fx_index`	Integer >= 0	FX selectors	Resolves to the zero-based index of the instance FX of this ReaLearn instance.
`tcp_fx_indexes`	Array of integers >= 0	FX selectors	Lets you access the indexes of FXs in a track control panel. E.g. `tcp_fx_indexes[2]` will resolve to the index of the third FX displayed in the track control panel.
`tcp_fx_parameter_indexes`	Array of integers >= 0	FX parameter selectors	Lets you access the indexes of FX parameters in a track control panel. E.g. `selected_fx_parameter_indexes[2]` will resolve to the index of the third FX parameter displayed in the track control panel. This only makes sense if used in conjunction with `tcp_fx_indexes`.

Example 32. Simple example

```
p1 * 99
```

- Will point to track with index 0 (first track) if Compartment parameter 1 is set to the minimum and to track with index 99 (= track number 100) if it's set to the maximum.

- If you use a formula like that, you should make sure that Compartment parameter 1 is controlled with a step size that allows for exactly 100 different values. This conforms to ReaLearn's default step size 0.01 = 1%.
- Since ReaLearn 2.13, this is easier because it adds support for integer parameters:
  - Set the Value count of the parameter to 100
  - You can now treat the parameter in the formula as an integer (just `p1` instead of `p1 * 99`).
  - Most importantly, ReaLearn will take care of using the correct step size automatically when setting up a mapping for controlling that parameter.

Example 33. More complex example

```
p1 * 3 * 100 + p2 * 99
```

This will treat Compartment parameter 1 as a kind of bank selector that allows you to choose between exactly 4 banks (0, 1, 2, 3) of 100 tracks each. Compartment parameter 2 will select the track number within the bank. You see, this is very flexible.

Additional object selectors for tracks

<This> selector

Track which hosts this ReaLearn instance. If ReaLearn is on the monitoring FX chain, this resolves to the master track of the current project.

<Selected> selector

Currently selected track. If multiple tracks are selected, refers only to the first one.

<All selected> selector

All currently selected tracks. This makes track targets (not FX target and not send targets) do their job on *all* selected tracks. The feedback value always corresponds to the highest value among all selected tracks.

If you select many tracks, things can become quite slow!

<Master> selector

Master track of the project which hosts this ReaLearn instance.

- If ReaLearn is on the monitoring FX chain, this resolves to the master track of the current project.
- If you don't have ReaLearn on the monitoring FX chain, but you want to control an FX on the monitoring FX chain, this option is the right choice as well. Make sure to enable the "Monitoring FX" checkbox.

All named selector

Allows you to use wildcards (see Named selector) to make track targets do their thing on all matching tracks instead of only the first one.

At TCP position selector

Like At position selector but just considers tracks that are visible in the track control panel.

At MCP position selector

Like At position selector but just considers tracks that are visible in the mixer control panel.

Dynamic (TCP) selector

Like Dynamic selector but the result should be an index counting only tracks visible in the track control panel.

Dynamic (MCP) selector

Like Dynamic selector but the result should be an index counting only tracks visible in the mixer control panel.

By ID or name (legacy) selector

This lets you refer to a track by its unique ID and name as fallback. This was the default behavior for ReaLearn versions up to 1.11.0 and is just kept for compatibility reasons.

This selector is deprecated! You shouldn't use it anymore.

Additional target selectors for FXs

<This> selector

Always points to the own ReaLearn FX Instance. Perfect for changing own parameters, e.g. for usage of the conditional activation or Dynamic selector features (especially important if you want to create reusable presets that make use of these features).

Focused selector

Currently or last focused FX. *Track* and *Input FX* settings are ignored.

Particular selector

Lets you pick a specific FX in the FX chain. Renaming the FX or moving it within the FX chain is fine - ReaLearn will still keep controlling exactly this FX. Please note that this only makes sense if you address the containing track using Particular selector as well.

Named selector

Lets you address the FX by its name in the FX chain. Just as with tracks, you can use wildcards to have a blurry search.

All named selector

Allows you to use wildcards (see Named selector) to make FX targets do their thing on all matching FX instances instead of only the first one.

By ID or position (legacy) selector

This refers to the FX by its unique ID with its position as fallback. This was the default behavior for ReaLearn versions up to 1.11.0 and is just kept for compatibility reasons.

This selector is deprecated! You shouldn't use it anymore.

Sticky selectors

We call object selectors *sticky* if they refer to a particular object (e.g. a track).

Sticky selectors

`<Master>`, `<This>`, `Particular`

Non-sticky selectors

`<Dynamic>`, `<Focused>`, `<Selected>`, `<Unit>`, `<All selected>`, `Named`, `All named`, `At position`, `From Playtime column`

Target property

Targets can expose properties, which you can use for Text feedback: Text expression or Dynamic feedback: Lua script.

Which properties are available, depends very much on the type of the target type.

There are some properties which are available for (almost) any target (for very target-specific properties, please look up the corresponding target in Targets):

Table 1. Common target properties

Name	Type	Description
`global.realearn.time`	Decimal	Time in milliseconds since ReaLearn has been loaded (the first instance).
`mapping.name`	String	Name of the mapping. Contains the explicitly assigned mapping name, never an automatically generated one.

Name	Type	Description
`target.text_value`	String	Short text representing the current target value, including a possible unit. If the target value can be represented using some kind of name, this name is preferred over a possibly alternative numeric representation. Example: Let's assume the 4th track in our project is called "Guitar" and the mapping target is Target "Project: Browse tracks". Then `target.text_value` contains the text *Guitar*, not the text *4*. This is the default value shown if textual feedback is enabled and the textual feedback expression is empty.
`target.available`	Boolean	A boolean value indicating whether the target is currently available or not. Most targets that are *active* are also *available*. But some targets can be *active* and *unavailable*. Example: Target "Pot: Preview preset" is not *available* if no preview is available for the preset currently selected in Pot browser. But the target is still considered *active* in this case! Usually used together with Dynamic feedback: Lua script, for example in order to display different things on displays depending on the target's availability.
`target.discrete_value`	Integer	The current target value as zero-based integer. This only works for discrete targets.
`target.discrete_value_count`	Integer	The number of possible values in the current target. This only works for discrete targets.
`target.numeric_value`	Decimal	The current target value as a "human-friendly" number without its unit. The purpose of this placeholder is to allow for more freedom in formatting numerical target values than when using `target.text_value`. This can be done using Dynamic feedback: Lua script.
`target.numeric_value.unit`	String	Contains the unit of `target.numeric_value` (e.g. *dB*).

Name	Type	Description
`target.normalized_value`	Decimal	The current target value as normalized number, that is, a value between 0.0 and 1.0 (the so-called unit interval). You can think of this number as a percentage, and indeed, it's currently always formatted as percentage. This value is available for most targets and good if you need a totally uniform and predictable representation of the target value that doesn't differ between target types. By default, this number is formatted as percentage *without* the percent sign. Future versions of ReaLearn might offer user-defined formatting. This will also be the preferred form to format on/off states in a custom way (where 0% represents *off*).
`target.type.name`	String	Short name representing the type of the mapping target.
`target.type.long_name`	String	Long name representing the type of the mapping target.
`target.track.index`	Integer	Zero-based index of the first resolved target track (if supported).
`target.track.name`	String	Name of the first resolved target track (if supported).
`target.track.color`	Color	Custom color of the first resolved target track (if supported).
`target.fx.index`	Integer	Zero-based index of the first resolved target FX (if supported).
`target.fx.name`	String	Name of the first resolved target FX (if supported).
`target.route.index`	Integer	Zero-based index of the first resolved target send/receive/output (if supported).
`target.route.name`	String	Name of the first resolved target send/receive/output (if supported).

Target value polling

Target value polling makes ReaLearn query the current value of a target every few milliseconds as part of the main application loop in order to send up-to-date feedback to your controller at all times.

Target value polling is not necessary for most targets because usually ReaLearn takes advantage of REAPER's internal notification system to get notified about target value changes (which is good for performance). For the few targets for which it is, this option is enabled by default in order to give you the best feedback experience out-of-the-box.

Remarks:

- For most targets that support polling, if you disable polling, automatic feedback for that target will simply stop working. This means you will not receive up-to-date feedback anymore whenever you change the target value within REAPER itself (not using ReaLearn).
- The Target "FX parameter: Set value" is an exception. Automatic feedback will still work, even without *Poll for feedback* enabled. But in the following corner cases it might not:
  - If the FX is on the monitoring FX chain.
  - If you change a preset from within the FX GUI.

Target activation condition

Targets can have activation conditions as well. They are very specific to the type of the target.

Example 34. Typical target conditions

- Track must be selected checkbox
- FX must have focus checkbox

Continuous vs. discrete value range

ReaLearn targets can have a *continuous* or *discrete* value range.

Continuous value range

A *continuous* value range is a range of arbitrary floating point numbers between 0.0 and 1.0. You can also think of them as *percentages* between 0.0% and 100.0%. Continuous value ranges don't have steps.

Example 35. Some targets with a continuous value range.

- Target "Track: Set volume"
- Target "Project: Set tempo"

Discrete value range

A *discrete* value range is a range of integers, e.g. 0 to 9. That would be 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. That's it! No other values are possible. Discrete value ranges have steps.

Example 36. Some targets with a discrete value range.

- Target "FX: Browse presets"
- Target "Project: Browse tracks"

Group properties

A group doesn't just have a name. It has a few properties that are also available for mappings and

thus affect all mappings in that group:

Tags

Tags defined in the group will be merged with the tags in each mapping.

Control/feedback-enabled

Lets you switch off control and/or feedback off for all mappings in that group.

Activation condition

The activation condition that you provide here is combined with the one that you provide in the mapping. Only if both, the group activation conditions and the mapping activation condition are satisfied, the corresponding mapping will be active. Read more about conditional activation in the section about the Mapping panel.

Best practices

Setting input and output

1. *Prefer output to a specific device over MIDI: <FX output>!*

   It's usually better to select a specific output device because sending MIDI feedback to the FX output has drawbacks.

   First, it doesn't participate in ReaLearn's multi-unit feedback orchestration. That means you might experience misbehaving LEDs/faders/displays when using multiple units.

   Second, it doesn't work if ReaLearn FX is suspended, e.g. in the following cases:

   - ReaLearn FX is disabled.
   - Project is paused and ReaLearn track is not armed.
   - ReaLearn FX is on input FX chain and track is not armed.

Using the controller compartment

The Controller compartment lets you describe a Controller simply by adding mappings. When you do that, each Controller mapping represents a Control element on your Controller, e.g. a Momentary button or Fader.

Describing your controller is optional but it brings benefits:

- You can use the Projection feature to project your controller mapping to a smartphone or tablet (watch video).
- You can use controller presets either built-in ones or those made by other users … and thereby save precious setup time. Or you can contribute them yourself!
- You can make your main mappings independent of the actually used Controller. This is done using virtual sources and virtual targets.
- It allows you to give your knobs, buttons etc. descriptive and friendly names instead of just e.g. "CC 15".
- You don't need to learn your control elements> again and again. Although the process of learning an element is easy in ReaLearn, it can take some time in case the <<further-concepts::source:::midi-source-character is not guessed correctly. Just do it once and be done with it!

If you want to make ReaLearn "learn" about your nice controller device, all you need to do is to create a suitable controller mapping for each of its control elements.

Let's first look at the "slow" way to do this - adding and editing each controller mapping one by one:

1. Press the Add one button.
2. Learn the Source by pressing the Learn source button and touching the control element.

3. Press the Edit button.
4. Enter a descriptive name for the Control element.

This name will appear in many places so you want it to be short, clear and unique!

5. Assign a unique Virtual target.
    - At this point we don't want to assign a Real target yet. The point of controller presets is to make them as reusable as possible, that's why we choose a Virtual target.
    - In the *Category* dropdown, choose *Virtual*.
    - As *Type*, choose Virtual control element type if your control element is a sort of button (something which you can press) or Multi in all other cases.
    - Use for each Control element a unique combination of Virtual control element type and Virtual control element ID, starting with number **1** and counting.

It's okay and desired to have one control element mapped to "Multi 1" and one to "Button 1".

    - Just imagine the "8 generic knobs + 8 generic buttons" layout which is typical for lots of popular controllers. You can easily model that by assigning 8 multis and 8 buttons.
    - Maybe you have realized that the Glue section is available for controller mappings as well! That opens up all kinds of possibilities. You could for example restrict the target range for a certain control element. Or make an encoder generally slower or faster. Or you could simulate a rotary encoder by making two buttons on your controller act as -/+ buttons emitting relative values. This is possible by mapping them to the same Virtual control element in Incremental button mode.

Before you go ahead and do that for each control element, you might want to check out what this is good for: Navigate back to the Main compartment, learn the Source of some Main mapping and touch the Control element that you have just mapped: Take note how ReaLearn will assign a Virtual source this time, not a MIDI source! It will also display the name of the Virtual control element as source label.

Now, let's say at some point you swap your Controller with another one that has a similar layout, all you need to do is to switch the Controller preset and you are golden! You have decoupled your Main mapping from the actual Controller. Plus, you can now take full advantage of the Projection feature.

All of this might be a bit of an effort, but it's well worth it! Plus, there's a way to do this *a lot* faster by using *batch learning*:

1. Press the Learn many button.
2. Choose whether you want to learn all the Multi elements on your Controller or all the Virtual control element type elements.
3. Simply touch all relevant control elements in the desired order. ReaLearn will take care of automatically incrementing the Virtual control element ID.

4. Press **[Stop]**.
5. Done!
    - At this point it's recommended to recheck the learned mappings.
    - ReaLearn's MIDI source character detection for MIDI CCs is naturally just a guess, so it can be wrong. If so, just adjust the character in the corresponding Mapping panel.

You can share your preset with other users by sending them to info@helgoboss.org. I will add it to this list.

Naming compartment parameters

Because ReaLearn's compartment parameters are freely assignable, they have very generic names by default. However, as soon as you give them meaning by using them in a specific way, it can be very helpful to give them a name by using the Compartment parameters menu.

Troubleshooting Luau import

The way Luau import works in ReaLearn is:

1. ReaLearn attempts to execute the Luau script in the clipboard.
2. ReaLearn attempts to interpret the returned value as ReaLearn API object.
3. ReaLearn loads the API object

If step 1 fails, ReaLearn displays an error messages that hopefully contains a line number. If step 2 fails, ReaLearn shows a validation error message.

If importing Luau code fails and the displayed error message is not helpful, you can try Dry-run Lua script from clipboard. This action enables you to just execute step 1 and see the "expanded" result. This can help to make sense of a possible validation error message in step 2.

Appendix A: REAPER actions

ReaLearn provides some REAPER actions which become available as soon as at least one instance of ReaLearn is loaded. It can be useful to put a ReaLearn instance on REAPER's monitoring FX chain in order to have access to those actions at all times.

In order to find these actions, open REAPER's *Actions* menu, choose *Show action list…* and simply search for `realearn`. The most important actions:

ReaLearn: Find first mapping by source

This action will ask you to touch some control element. As soon as you touch a control element which is mapped, it will open the mapping panel for the corresponding mapping. It will search within all ReaLearn instances/units loaded in your current project as well as the ones on the monitoring FX chain.

ReaLearn: Find first mapping by target

This action is similar to *Find first mapping by source*. It asks you to touch some (learnable) REAPER parameter. As soon as you touch one that is mapped, it will open its mapping panel.

ReaLearn: Learn single mapping (reassigning source)

Asks you to touch a control element and target and adds a new mapping in the first ReaLearn unit that it encounters. It prefers units in the current project over monitoring FX. It automatically chooses the unit with the correct MIDI/OSC input. If there's a unit which already has that source assigned, it will be reassigned to the new target that you touched.

ReaLearn: Learn single mapping (reassigning source) and open it

Like *Learn single mapping* but additionally opens the mapping panel after having learned the mapping. This is great for subsequent fine-tuning.

ReaLearn: Learn source for last touched target (reassigning target)

This behaves similar to REAPER's built-in MIDI learn in that it always relates to the target that has been touched last.

ReaLearn: Send feedback for all instances

Makes each ReaLearn instance/unit in all project tabs send feedback for all mappings. That shouldn't be necessary most of the time because ReaLearn usually sends feedback automatically, but there are situations when it might come in handy.

Configuration files

ReaLearn creates and/or reads a few files in REAPER's resource directory.

`Data/helgoboss`
: Directory which contains data such as presets or resources that need to be distributed via ReaPack

`Data/helgoboss/auto-load-configs/fx.json`
: Contains global FX-to-preset links, see Auto-load

`Data/helgoboss/archives`
: Directory which contains archives e.g. the compressed app, distributed via ReaPack

`Data/helgoboss/doc`
: Contains offline documentation, e.g. this guide as PDF

`Data/helgoboss/presets/controller`
: Contains preset for the controller compartment

`Data/helgoboss/presets/main`
: Contains preset for the main compartment

`Helgoboss`
: Directory which contains rather personal or device-specific data, not touched via ReaPack

`licensing.json`
: Contains license keys

`Helgoboss/App`
: Contains the uncompressed App, if installed

`Helgoboss/Pot/previews`
: Directory which contains previews recorded by Pot Browser

`Helgoboss/ReaLearn/osc.json`
: Global OSC device configurations, see Manage OSC devices

`Helgoboss/ReaLearn/realearn.ini`
: Very basic global configuration, currently mainly regarding ReaLearn's built-in server.

  Currently supported properties (subject to change): `server_enabled`, `server_http_port`, `server_https_port`, `server_grpc_port`, `companion_web_app_url`

`Helgoboss/Server/certificates`
: Contains a list of certificates and corresponding private keys in order to allow encrypted communication with ReaLearn Companion and App.

Design decisions

Expert level!

This section sheds a bit light about why some features in ReaLearn exist and why they were built as they are. Just in case you are interested.

Advanced settings via YAML

The Advanced settings dialog makes it possible to configure a few advanced mapping settings by entering text in the YAML configuration language.

Deciding for textual configuration and YAML in particular was a conscious decision with the goal to provide a developer-friendly framework for rapidly extending ReaLearn with advanced features that don't urgently need a graphical user interface.

1. *Why ask the user to enter text instead of providing a convenient graphical user interface?*
    - That's mostly a tradeoff due to the fact that my time available for developing ReaLearn is limited.
    - It's much work to develop a graphical user interface for every feature. In fact, programming the user interface often takes most of the time whereas implementing the actual logic is not that much effort.
    - It's true that some sorts of functionality really benefit from having a fancy graphical user interface. But there's also functionality for which having it is not too important, e.g. functionality that is of configurational nature and not used that often.
    - Also, one of ReaLearn's goals is to give power users and programmers extra powers. Textual configuration can be more powerful in many situations once the user knows how to go about it.
2. *Why YAML?*
    - YAML has the advantage of being popular among programmers, widely supported, highly structured and relatively well readable/writable by both humans and machines.
    - Many text editors offer built-in support for editing YAML.
    - Makes it easy to provide data for even very complex features.
3. *Why not a scripting language?*
    - Using a scripting language would spoil any future possibility to add a graphical user interface on top of some of the functionality.
    - It wouldn't allow ReaLearn to apply future optimizations and improvements. ReaLearn is rather declarative in nature and a scripting language would destroy this quality.
    - It's hard to come up with a stable and good API.
    - It's harder to use than a configuration language.
4. *Why don't you save the text, just the structure?*
    - Mostly because saving just the structure makes the entered data become a natural part of

ReaLearn's main preset format (JSON).

- Once we would start saving the actual text, it would be hard to go back.