ReaLearn User Guide

Last update of text:	`2020-09-10 (v1.11.0)`
Last update of relevant screenshots:	`2020-09-10 (v1.11.0)`

1 Introduction

1.1 What is ReaLearn?

Probably you know already that ReaLearn is a kind of improved MIDI learn for REAPER. But what is it exactly? Let me put it this way:

> ReaLearn is an instrument. It allows you to take whatever MIDI controller you have, be it a keyboard or some fader box, plug it in and play ... but instead of playing notes, you "play" REAPER itself!
>
> And because ReaLearn supports MIDI feedback, you can also do the reverse: Let REAPER "play" your controller.

While this is still pretty vague, I think it captures the essence of ReaLearn. From a technical viewpoint it's a VSTi plug-in, so it is an instrument, quite literally. That's one thing that sets it immediately apart from the more conventional control surface feature in REAPER and 3rd-party efforts such as CSI or DrivenByMoss. The goal of the latter-mentioned is to equip REAPER with out-of-the-box support for specific controllers, typically dedicated DAW controllers such as Mackie MCU that are tailored to control a DAW just like a hardware mixer. And I suppose they do a pretty good job at that.

ReaLearn's goal is quite different: It enables you to instantly map control elements to REAPER parameters and save the resulting mappings as part of your project. Never again you need to pollute your global control mappings just for the needs of one project.

The usual ReaLearn workflow goes like this:

1. Hit "Learn source" and touch some knob on your controller.
2. Hit "Learn target" and touch some target parameter.
3. Done.

The result is a mapping that you can customize as you desire, for example by setting a target value range. All of that with MIDI feedback support, which was previously only available in the less dynamic, more global control surface world.

Summary: *ReaLearn is a sort of instrument for controlling REAPER.*

1.2 Usage scenarios

Ultimately, ReaLearn gains whatever purpose you can come up with. Because it is a VSTi plug-in and provides many MIDI routing options, it's very flexible in how it can be used. You can "inject" it wherever you want or need it (limitation: using it in a take FX chain is not possible yet):

- **Input FX chain for live-only use:** Put it on a track's input FX chain in order to use it only for incoming "live" MIDI and let it control a parameter of an effect that's on the normal FX chain, right below a synthesizer. It will be active only if the track is armed for recording.
- **Grid controller for song switching:** Use some grid controller like the AKAI APC Key 25 to arm/disarm various tracks (effectively enabling/disabling certain sound setups) by pressing the grid buttons - with the LEDs of the buttons indicating which setup is currently active.
- **Combination with other MIDI FX for interesting effects:** Slap it on a track FX chain, right between a MIDI arpeggiator and a synthesizer to arpeggiate the cutoff parameter of that synthesizer.
- **Monitoring FX for project-spanning setups:** Put it on the monitoring FX chain to have some control mappings available globally in all projects (similar to conventional control surface stuff).
- **Unusual settings for experimental stuff:** Create a track volume mapping with only feedback turned on. Choose "<FX output>" as MIDI feedback output and play the synthesizer one position below in the FX chain by moving the track volume slider (whatever that might be good for ...).
- **Rotary encoders for avoiding parameter jumps:** How about a refreshingly "normal" use case? Let your rotary endless encoder control a track send volume without parameter jumps and restrict the value range to volumes below 0dB.
- **Presets for easy reuse:** Save a bunch of commonly used mappings globally as FX presets.
- **Combination of multiple instances:** Use one ReaLearn instance to arm or disarm tracks that contain other ReaLearn instances to enable/disable different mapping groups. Group mappings and activate/deactivate them group-wise simply by instantiating multiple ReaLearn instances and enabling/disabling them as desired in the FX chain window.

... the possibilities are endless. It's all up to you! Use your creativity.

All of that makes ReaLearn especially well-suited for performers, people who use REAPER as a platform for live playing. It might be less interesting to people who use REAPER for arranging, mixing and mastering only. But even so, as long as you have some general-purpose MIDI controller and you want a fine-tuned mapping to DAW parameters of all sorts, give ReaLearn a try. It might be just what you need. More so if the controller supports feedback (e.g. motorized faders or LEDs).

Summary: *ReaLearn is tailored to usage scenarios typically desired by performers.*

1.3 Videos

If you want to get a first impression of ReaLearn, a video is surely a good way. I've never made a ReaLearn video myself but other users luckily did. Here a short, non-complete list:

- How To: ReaLearn and MIDI Controller for Track Sends in REAPER - Tutorial
- using ReaLearn to assign MIDI controllers to (VST) plugin parameters in Cockos Reaper
- MIDI Controller Feedback in REAPER with ReaLearn and LBX SmartKnobs - Tutorial

Keep in mind that the mentioned videos still use older versions of ReaLearn.

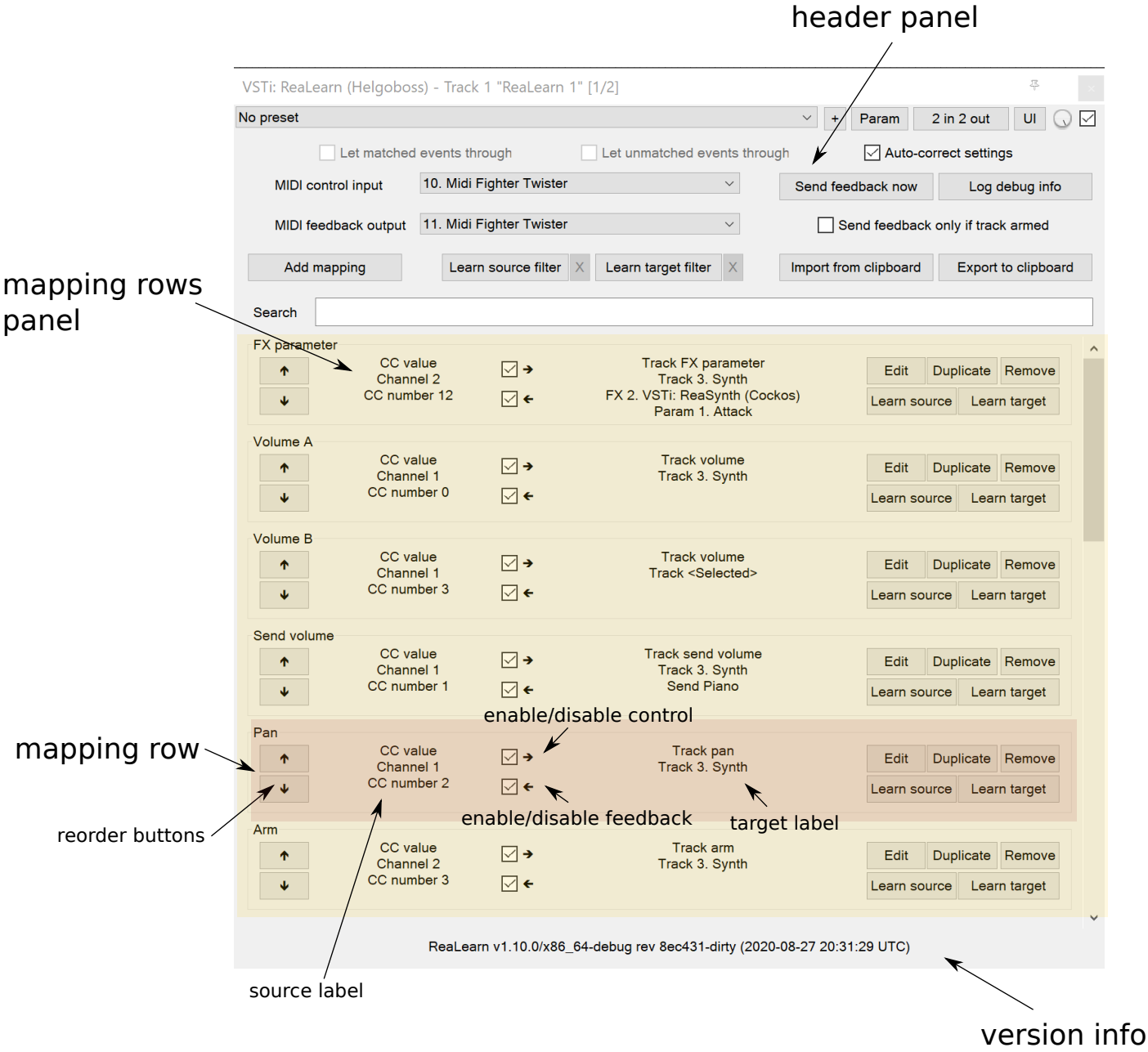
2 Basics

2.1 Control

After installing ReaLearn, you can fire it up just like any other VST instrument in REAPER: By adding it to an FX chain.

1. Right click in the track header area and choose "Insert virtual instrument on new track..."
2. Choose "VSTi: ReaLearn (Helgoboss)"

After that you should see ReaLearn's main panel (unlike this screenshot, it wouldn't contain any mappings yet):



On the very top you see the *header panel* for changing settings or executing actions that affect this complete instance of ReaLearn. Below that there's the *mapping rows panel* which displays all mappings in this instance of ReaLearn. There can be very

many of them. I've heard from users who use hundreds. On the very bottom you see some information about the version of ReaLearn that you are running. It's important to include this information in bug reports.

2.1.1 Adding a mapping

Let's see how to add and use our first mapping:

1. Press the "Add mapping" button
    - A new mapping called "1" should appear in the mapping rows panel.
2. Press the "Learn source" button of that new mapping.
    - Its label will change to "Stop".
3. Touch some control element on your MIDI controller (knob, encoder, fader, button, key, pitch bend, mod wheel, ...). For this example it's best to use something continuous, not a button or key.
    - If your MIDI is set up correctly, the button label should jump back to "Learn source" and the touched control element should appear in the *source label*. See below if this doesn't happen.
4. Press the "Learn target" button.
    - Its label will change to "Stop".
5. Touch the volume fader of your newly created REAPER track.
    - The button label should jump back to "Learn target" and "Track volume" should appear in the *target label*.
6. Now you should be able to control the touched target with your control element.

2.1.2 Troubleshooting

If REAPER crashes when scanning for plug-ins and the crash message shows something like `reaper_host64` or `reaper_host32`, you either have a 32/64-bit version mismatch or you have "Preferences → Plug-ins → Compatibility → VST bridging/firewalling" set to "In separate plug-in process" or "In dedicated process per plug-in". Please see the installation instructions on the project website for hints how to fix this. In future, ReaLearn hopefully will handle this situation more gracefully.

If the label remains at "Stop" at step 3, you need to have a look at your MIDI setup.

- Make sure **Enable input from this device** is checked for your controller MIDI input device in the REAPER preferences.
    - Please note: *Enable input for control messages* is totally irrelevant for ReaLearn. This is only used for REAPER's built-in MIDI learn, which uses the so-called *control MIDI path*. ReaLearn on the other hand - when *MIDI control input* is set to "<FX input>" - uses the regular *track MIDI path*, which is one reason why it is so flexible.
- Make sure your audio hardware is not stuck (playback in REAPER should work).
- Make sure the track is armed for recording and has the appropriate MIDI device input.
- Make sure your controller is in MIDI mode.
    - Some controllers, especially DAW controllers, are able to work with several protocols (MCU, HUI, ...).

- Consult your controller's manual and take the necessary steps to put it into something like a "generic MIDI" mode.
- Example: Presonus Faderport

When you read this the first time, you might get the impression that this is a lot of work for setting up one simple control mapping. It's not. Learning mappings is a matter of a few secs after you got the hang of it. ReaLearn also registers a REAPER action "Learn source for last touched target" which might further speed up this step. More about that later.

At this point: Congratulations! You have successfully made your first baby steps with ReaLearn.

2.1.3 Some words about routing

If you think that what we saw until now is not more than what REAPER's built-in MIDI learn already offers, I can't blame you. First, don't worry, there's more to come, this was just the beginning. Second, there *is* a difference. For some folks, this is an insignificant difference, for others it's a game changer, it depends on the usage scenario. The key to understand this difference is to understand the MIDI *routing*: We have just used the so-called *track MIDI path* to control a parameter in REAPER. This is different from REAPER's built-in MIDI learn, which uses the *control MIDI path*.

Using the track MIDI path means it's completely up to you to decide what MIDI messages flow into ReaLearn. You decide that by using REAPER's powerful routing capabilities. For example, you can simply "disable" the mapping by disarming your track, a feature that is very desireable if you use REAPER as live instrument. Or you can preprocess incoming MIDI (although that should rarely be necessary given ReaLearn's mapping customization possibilities).

Another thing worth to point out that is different from built-in MIDI learn is that we didn't use the action "Track: Set volume for track 01". Benefit: ReaLearn will let you control the volume of the track even if you move that track to another position. The track's position is irrelevant.

2.2 Feedback

In ReaLearn, every mapping has 2 directions: *control* (controller to REAPER) and *feedback* (REAPER to controller). So far we have talked about the *control* direction only: When you move a knob on your controller, something will happen in REAPER. But if your controller supports it, the other direction is possible, too!

Imagine that you used a MIDI-controllable motorized fader as control element to change the track volume. ReaLearn is capable of making that fader move whenever your track volume in REAPER changes - no matter if that change happens through automation or through dragging the fader with your mouse. Motorized faders are quite fancy. Another form of feedback visualisation are rotary encoders with LEDs that indicate the current parameter value.

How to set this up? Often it's just a matter of choosing the correct feedback device:

1. Make sure **Enable output to this device** is checked for your controller MIDI output device in the REAPER preferences.
2. In ReaLearn's header panel, select your controller as *MIDI feedback output*.

That should be it!

If it doesn't work and you have ruled out MIDI connection issues, here are some possible causes:

1. **Your controller is not capable of feedback via generic MIDI messages.**
    - Some controllers *do* support feedback, but not via MIDI. Or via MIDI but using some custom sys-ex protocol instead of generic MIDI messages.
    - In this case, ReaLearn can't help you. Reverse engineering custom protocols is out of ReaLearn's scope.
    - Recommendation: Maybe you are able to find some bridge driver for your controller that is capable of translating generic MIDI messages to the proprietary protocol. Then it could work.
    - Examples: Akai Advance keyboards, Native Instruments Kontrol keyboards, Arturia MiniLab
2. **Your controller has multiple modes and currently is in the wrong one.**
    - Some controllers, especially DAW controllers, are able to work with several protocols.
    - Recommendation: Consult your controller's manual and take the necessary steps to put it into something like a "generic MIDI" mode.
    - Example: Presonus Faderport
3. **Your controller expects feedback via different MIDI messages.**
    - Usually, controllers with feedback support are kind of symmetric. Here's an example what I mean by that: Let's assume your motorized fader *emits* CC 18 MIDI messages when you move it. That same motorized fader starts to move when it *receives* CC 18 MIDI messages (messages of exactly the same type). That's what I call symmetric. E.g. it's not symmetric if it emits CC 18 but reacts when receiving CC 19.
    - ReaLearn assumes that your controller is symmetric. If it's not, you will observe non-working or mixed-up feedback.
    - Recommendation: Consult your controller's manual and try to find out which MIDI messages need to be sent to the controller to deliver feedback to the control element in question. Then, split your mapping into two, making the first one a control-only and the second one a feedback-only mapping. Adjust the source of the feedback-only mapping accordingly. In the next section you'll learn how to do that.
    - Example: Presonus Faderport

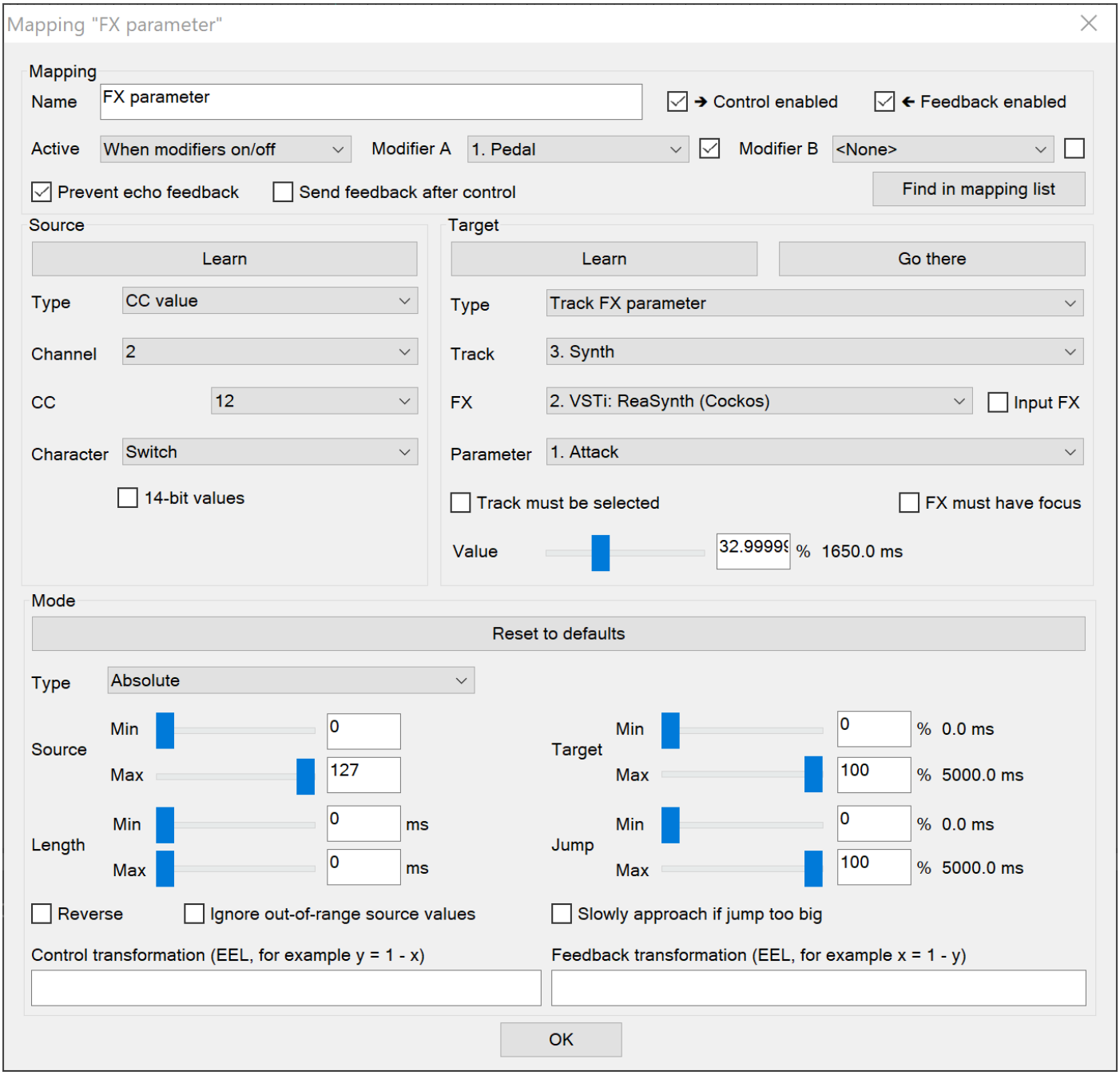
Personally, I've made good feedback experiences with the following controllers (but I haven't tried very many, so this is for sure a very incomplete list):

- DJ TechTools Midi Fighter Twister
- Akai APC Key 25
- Presonus Faderport

All hardware examples are provided to the best of my knowledge. If anything is incorrect or has changed in the meanwhile, please let me know!

2.3 Editing a mapping

When you press the *Edit* button of a mapping row, a so-called *mapping panel* appears, which lets you look at the corresponding mapping in detail and modify it:



This panel has 4 sections:

- **Mapping:** Allows to change the name and other general settings related to this mapping.
- **Source:** Allows to edit the *source* of the mapping. In most cases, a source represents a particular control element on your controller (e.g. a fader).
- **Target:** Allows to edit the *target* of the mapping and optionally some target-related activation conditions. A target essentially is the parameter in REAPER that should be controlled.
- **Mode:** Allows to edit everything related to the *mode* of this mapping. A mode is the glue between source and target. It defines *how* incoming control values from the source should be applied to the target (and vice versa, if feedback is used). This is where it gets interesting. Unlike REAPER's built-in MIDI learn modes, ReaLearn modes are deeply customizable.

By design, source, mode and target are independent concepts in ReaLearn. They can be combined freely - although there are some combinations that don't make too much sense.

Changes in the mapping panel are applied immediately. Pressing the *OK* button just closes the panel.

Tip: It is possible to have up to 4 mapping panels open at the same time.

2.4 Controller setup

In order to get the most out of your controller in combination with ReaLearn, you should consider the following hints:

- Put your controller's buttons into momentary mode, *not* toggle mode.
- If you are in the lucky situation of owning a controller with endless rotary encoders, by all means, configure them to transmit relative values, not absolute ones!
    - Otherwise you can't take advantage of "Relative mode" features such as the "Step size" or "Speed" setting.

3 Reference

So far we've covered the basics. Now let's look into everything in detail.

3.1 Main panel

3.1.1 Header panel

- **MIDI control input:** By default, ReaLearn captures MIDI events from *<FX input>*, which consists of all MIDI messages that flow into this ReaLearn VSTi FX instance (= track MIDI path). Alternatively, ReaLearn can capture events directly from a MIDI hardware input. This dropdown lets you choose the corresponding MIDI input device. Be aware that this will only work if *Enable input from this device* is checked for the selected MIDI input device in REAPER's MIDI preferences.
- **MIDI feedback output:** Here you can choose if and where ReaLearn should send MIDI feedback. By default it's set to *<None>* for no feedback. If you want to enable feedback, pick a MIDI output device here. Keep in mind that *Enable output to this device* must be checked in REAPER's MIDI preferences. As an alternative, you can send feedback to *<FX output>*, which makes feedback MIDI events stream down to the next FX in the chain or to the track's MIDI output.
- **Let matched events through / Let unmatched events through:** These settings only apply if MIDI control input is set to *<FX input>*. By default, ReaLearn "eats" incoming MIDI events for which there's at least one enabled mapping source. In other words, it doesn't forward MIDI events which are used to control a target parameter. All other MIDI events are forwarded to ReaLearn's FX output. Use these checkboxes to change that behavior.
- **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. E.g. if you pick a source which emits relative increments,

it will automatically select a relative mode for you - because nothing else makes sense. This auto-correction is usually very helpful, especially if you *learn* a source instead of selecting it manually. Auto-correction is what gives *learning* a snappy user experience. If for some reason you want to disable auto-correction, this is your checkbox.

- **Send feedback only if track armed:** If MIDI control input is set to *<FX input>*, ReaLearn by default disables feedback if the track is not armed (unarming will naturally disable control, so disabling feedback is just consequent). However, if MIDI control input is set to a hardware device, the default is to *not* disable feedback when unarmed (same reasoning). You can override this behavior with this checkbox. At the moment, it can only be unchecked if ReaLearn is on the normal FX chain. If it's on the input FX chain, unarming naturally disables feedback because REAPER generally excludes input FX from audio/MIDI processing while a track is unarmed.
- **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.
- **Log debug info:** Logs some information about ReaLearn's internal state. Can be interesting for investigating bugs or understanding how this plug-in works.
- **Add mapping:** Inserts a default mapping at the end of the current mapping list.
- **Learn source filter:** If you work with many mappings and you have problems memorizing them, you will love this feature. When you press this button, ReaLearn will start listening to incoming MIDI 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. *Tip:* 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 "learn source filter" mode. Then just calm down and press *Stop*. It's easy to forget.
- **Learn target filter:** 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 *Learn source filter*, ReaLearn will automatically stop learning as soon as a target was touched. Press the **X** button to remove the filter and show all mappings again.
- **Export to clipboard / Import to clipboard:** Pressing the export button copies a *complete* dump of ReaLearn's current settings (including all mappings) to the clipboard. Pressing the import button does the opposite: It restores whatever ReaLearn dump is currently in the clipboard. This is a very powerful feature because the dump's data format is 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 your entrance ticket to batch editing. You can also use it for some very basic A/B testing (1. Press *Export to clipboard*, 2. change some settings and test them, 3. Restore the old settings by pressing *Import from clipboard*).
- **Search:** Enter some text here in order to display just mappings whose name matches the text.

3.1.2 Mapping row

- **Up / Down:** Use these buttons to move this mapping up or down the list.
- **→ / ←:** Use these checkboxes to enable/disable control and/or feedback for this mapping.
- **Edit:** Opens the mapping panel for this mapping.
- **Duplicate:** Creates a new mapping just like this one right below.
- **Remove:** Removes this mapping from the list.
- **Learn source:** Starts or stops learning the source of this mapping.
- **Learn target:** Starts or stops learning the target of this mapping.

3.2 Mapping panel

At this point it's important to understand some basics about how ReaLearn processes incoming control events. When there's an incoming control event that matches a particular source, one of the first things ReaLearn does is to normalize it to a so-called *control value*.

A control value can be either absolute or relative, depending on the source character:

- **Source emits absolute values (e.g. faders)**: The control value will be absolute, which means it's a 64-bit decimal number between 0.0 and 1.0. You can also think of it in terms of percentages: Something between 0% and 100%. 0% means the minimum possible value of the source has been emitted whereas 100% means the maximum.
- **Source emits relative values (e.g. rotary encoders)**: The control value will be relative, which means it's a positive or negative integer that reflects the amount of the increment or decrement. E.g. -2 means a decrement of 2.

After having translated the incoming event to a control value, ReaLearn feeds it to the mapping's *mode*. The mode 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 mapping's target character. The mode can even "eat" control values so that they don't arrive at the target at all.

Finally, ReaLearn converts the transformed control value into some target instruction (e.g. "set volume to -6.0 dB") and executes it.

Feedback (from REAPER to controller) 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.

3.2.1 Mapping

This section provides the following mapping-related settings and functions:

- **Name:** 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.
- **Control enabled / Feedback enabled:** Use these checkboxes to enable/disable control and/or feedback for this mapping.
- **Active:** This dropdown controls so-called conditional activation of mappings. See section below.
- **Prevent echo feedback:** This checkbox 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 *but at the same time* insist on controlling these LEDs themselves. According to users, some Behringer X-Touch Compact buttons exhibit this behavior, for example. This can lead to wrong LED states which don't reflect the actual state in REAPER. If this checkbox is not checked (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. However, if this checkbox is checked, 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"). By checking this checkbox, ReaLearn will send feedback whenever the target value change was caused by ReaLearn itself, which improves the situation at least a bit.
- **Find in mapping list:** Scrolls the mapping rows panel so that the corresponding mapping row for this mapping gets visible.

3.2.2 Conditional activation

Conditional activation allows you to dynamically enable or disable this mapping based on the state of ReaLearn's own plug-in parameters. This is a powerful feature. It is especially practical if your controller has a limited amount of control elements and you want to give control elements several responsibilities. It let's 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!"
- "I want to have two buttons for switching between different programs where each program represents a group of mappings."

There are 4 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 program selected:** Allows you to step through different groups of mappings
- **When EEL result > 0:** Let a formula decide (total freedom)

For details, see below.

At this occasion some words about ReaLearn's plug-in parameters. ReaLearn itself isn't just able to control parameters of other FX, it also offers FX parameters itself. At the moment these are "Parameter 1" to "Parameter 20". You can control them just like parameters in other FX: Via automation envelopes, via track controls, via REAPER's own MIDI 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 conditional activation. In future, ReaLearn will provide additional ways to make use of parameters.

3.2.2.1 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 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.

3.2.2.2 When program selected

You can tell ReaLearn to only activate your mapping if a certain parameter has a particular value. The certain parameter is called "Bank" and the particular value is called "Program". Why? Let's assume you mapped 2 buttons "Previous" and "Next" to increase/decrease the value of the "Bank" parameter (by using "Relative" mode, you

will learn how to do that further below). And you have multiple mappings where each one uses "When program selected" with the same "Bank" parameter but a different "Program". Then the result is that you can press "Previous" and "Next" and it will switch between different mappings (programs) within that bank. If you assign the same "Program" to multiple mappings, it's like putting those mapping 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:

- **Navigate between programs using a rotary encoder:** Just map the rotary encoder to the "Bank" parameter ("Relative" mode) and restrict the target range as desired.
- **Activate each program with a separate button:** Map each button to the "Bank" parameter ("Absolute" mode) 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 program 1 and pressing button 2 will activate program 2.

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 "Track FX enable" target 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 "Track FX preset" target in order to navigate between 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 instance. A fixed assumption here is that each bank (parameter) consists of 100 programs. If this is too limiting for you, please use the EEL activation mode instead.

3.2.2.3 When EEL result > 0

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 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 `p20` 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.

3.2.2.4 Custom parameter names

There's a somewhat hidden possibility to give ReaLearn parameters more descriptive names (yes, not very convenient, hopefully future versions will improve on that):

1. Press *Export to clipboard* in the main panel.
2. Paste the result into a text editor of your choice.
3. You will see a property "parameters", e.g.

```
"parameters": {
  "0": {
    "value": 0.084
  }
}
```

4. Adjust it as you like, e.g.

```
"parameters": {
  "0": {
    "value": 0.084,
    "name": "Pedal"
  },
  "1": {
    "name": "Shift"
  }
}
```

5. Copy the complete text to the clipboard.
6. Press *Import from clipboard* in the main panel.

Parameter names are not global, they are always saved together with the REAPER project / FX preset / track template etc.

3.2.3 Source

As mentioned before, a source usually represents a single control element on your controller. Sources share the following common settings and functions:

- **Learn:** Starts or stops learning the source of this mapping.
- **Type:** Let's you choose the MIDI source type.
- **Channel:** Optionally restricts this source to messages from a certain MIDI channel. Only available for sources that emit MIDI channel messages.

All other UI elements in this section depend on the chosen source type.

3.2.3.1 CC value source

This source reacts to incoming MIDI control-change messages.

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

- **Character:** MIDI control-change messages (7-bit ones) 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 a single direction. The possible values are:
  - **Range element (knob, fader, etc.):** A control element that emits continuous absolute values. Examples: Faders, knobs, modulation wheel, pitch bend, ribbon controller.
  - **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.
    - Hint: There's no option "Button (toggle)" because ReaLearn can only take full control with momentary buttons. So make sure your controller buttons are in momentary mode! ReaLearn itself provides a toggle mode that is naturally more capable than your controller's built-in toggle mode.
  - **Encoder (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)
- **14-bit values:** 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.

3.2.3.2 Note velocity source

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:** 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).

3.2.3.3 Note number source

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.

3.2.3.4 Pitch wheel source

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%.

3.2.3.5 Channel after touch source

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

3.2.3.6 Program change source

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

3.2.3.7 (N)RPN value source

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.

- **Number:** The number of the registered or unregistered parameter-number message. This is a value between 0 and 16383.
- **RPN:** If unchecked, this source reacts to unregistered parameter-number messages (NRPN). If checked, it reacts to registered ones (RPN).
- **14-bit values:** If unchecked, this source reacts to (N)RPN messages with 7-bit resolution. If checked, it reacts to those with 14-bit resolution. In practice, this if often checked.

3.2.3.8 Polyphonic after touch source

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

- **Note:** Optionally restricts this source to messages with a certain note number.

3.2.3.9 MIDI clock tempo source

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 *Master tempo* target 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).

3.2.3.10 MIDI clock transport source

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:** The specific transport message to which this source should react.

3.2.4 Target

A target is a thing that is supposed to be controlled. The following settings and functions are shared among all targets:

- **Learn:** Starts or stops learning the target of this mapping.
- **Go there:** If applicable, pressing this button 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.
- **Type:** Let's you choose the target type.
- **Value:** Reflects the current value of this mapping target and lets you change it.

Only available for targets that are associated with a particular REAPER track:

- **Track:** The track associated with this target. In addition to concrete tracks, the following options are possible:
  - **<This>**: 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>**: Currently selected track.
  - **<Master track>**: 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.
- **Track must be selected:** If checked, this mapping will be active only if the track set in *Track* is currently selected. Of course, this doesn't have any effect if latter is

<Selected>.

Only available for targets associated with a particular track send:

- **Send:** The (outgoing) send (to another track) associated with this target.

Only available for targets associated with a particular FX instance:

- **FX:** The FX instance associated with this target.
- **Input FX:** 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.
- **FX must have focus:** If checked, this mapping will be active only if the FX instance set in *FX* 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.

All other UI elements in this section depend on the chosen target type.

3.2.4.1 Action target

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

- **Pick:** Opens REAPER's action dialog so you can select the desired 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:** 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.
  - **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.

3.2.4.2 Track FX parameter target

Sets the value of a particular track FX parameter.

- **Parameter:** The parameter to be controlled.

3.2.4.3 Track volume target

Sets the track's volume.

3.2.4.4 Track send volume target

Sets the track send's volume.

3.2.4.5 Track pan target

Sets the track's pan value.

3.2.4.6 Track arm target

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 the *Track selection* target instead.

3.2.4.7 Track selection target

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

- **Select exclusively:** If unchecked, this leaves all other tracks' selection state untouched. If checked, this unselects all other tracks when selecting this track.

3.2.4.8 Track mute target

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

3.2.4.9 Track solo target

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

3.2.4.10 Track send pan target

Sets the track send's pan value.

3.2.4.11 Master tempo target

Sets REAPER's master tempo.

3.2.4.12 Master playrate target

Sets REAPER's master playrate.

3.2.4.13 Track FX enable target

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

3.2.4.14 Track FX preset target

Steps through FX presets.

3.2.4.15 Selected track target

Steps through tracks.

3.2.4.16 Track FX all enable

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

3.2.4.17 Transport

Invokes a transport-related action.

- **Action:** 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.
  - **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.

3.2.5 Mode

As mentioned before, a mode is the glue between a source and a target. Modes share the following common settings and functions:

- **Reset to defaults:** Resets the settings for the chosen mode type to some sensible defaults.

- **Type:** Let's you choose the mode type.

- **Source Min/Max:** The observed range of absolute source control values. By restricting that range, you basically tell ReaLearn to react only to a sub range of a control element, e.g. only the upper half of a fader or only the lower velocity layer of a key press. In relative mode, this only has an effect on absolute source control values, not on relative ones. This range also determines the minimum and maximum feedback value.

- **Target Min/Max:** The controlled range of absolute target values. This enables you to "squeeze" target values into a specific value range. E.g. if you set this to "-6 dB to 0 dB" for a *Track volume* target, the volume will always stay within that dB range if controlled via this mapping. This wouldn't prevent the volume from exceeding that range if changed e.g. in REAPER itself. This setting applies to targets which are controlled via absolute control values (= all targets with the exception of the "Action target" if invocation type is *Relative*).

- **Out of range:** This determines ReaLearn's behavior if the source value is not within "Source Min/Max" or the target value not within "Target Min/Max". There are these variants:

Control direction (absolute mode only)	Feedback direction

	Control direction (absolute mode only)	Feedback direction
Min or max	If the source value is < "Source Min", ReaLearn will behave as if "Source Min" was received (or 0% if min = max).	If the target value is < "Target Min", ReaLearn will behave as if "Target Min" was detected (or 0% if min = max).
	If the source value is > "Source Max", ReaLearn will behave as if "Source Max" was received (or 100% if min = max).	If the target value is > "Target Max", ReaLearn will behave as if "Target Max" was detected (or 100% if min = max).
Min	ReaLearn will behave as if "Source Min" was received (or 0% if min = max).	ReaLearn will behave as if "Target Min" was detected (or 0% if min = max).
Ignore	Target value won't be touched.	No feedback will be sent.

All other UI elements in this section depend on the chosen mode type.

3.2.5.1 Absolute (for range elements and buttons)

Absolute mode takes and optionally transforms absolute source control values. *Absolute* means that the current target value is irrelevant and the target will just be set to whatever control value is coming in (potentially transformed). If incoming source control values are relative, they will be ignored.

- **Length Min/Max:** This decides how long a button needs to be pressed to have an effect. Obviously, this setting makes sense for button-like control elements only (keys, pads, buttons, ...), not for knobs or faders.
  - 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 *Length Min* to e.g. 1000 ms and *Length 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
- **Jump Min/Max:** 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. *Jump Min/Max* settings can be used to control jump behavior.
    - You can imagine the *Jump Max* setting as the maximum allowed parameter jump (distance between two target values). By default, jumps of up to 100% are allowed, which means things can get very jumpy. You can set this to a very low value, e.g. 5%. Then, when you move the fader, ReaLearn will do absolutely nothing *until* the fader comes very close to the current target value. This is called *pick up mode* in some DAWs (what an appropriate name!). Make sure to not set *Jump Max* too low, otherwise your target value might get stuck.
    - The *Jump Min* setting is more unconventional. If you raise *Jump Min*, this effectively enforces parameter jumps. It's like adjusting target values to a relative grid.
- **Slowly approach if jump too big:** If you combine *Jump Max* with enabling *Slowly approach if jump too big*, you gain a "Soft takeover" effect, as it is called in REAPER's built-in MIDI learn. In some other DAWs this is called "Scale mode". This is similar to "pick up" with the difference that the current target value will gradually "come your way". This results in pretty seemless target value adjustments but can feel weird at times because the target value can temporarily move in the opposite direction of the fader movement.
- **Reverse:** If checked, this inverses the direction of the change. E.g. the target value will decrease when moving the fader upward and increase when moving it downward.
- **Control transformation (EEL):** This feature definitely belongs in the "expert" category. It allows you to write a formula that transforms incoming control values before they are passed on to the target. 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. 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. Please note that the initial value of `y` is the current target value, so you can even "go relative" in absolute mode.
- **Feedback transformation (EEL):** This is like *Control transformation (EEL)* but used for translating a target value back to a source value for feedback purposes. It usually makes most sense if it's exactly the reverse of the control transformation. 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: `x = y * 2`.

3.2.5.2 Relative (for encoders and buttons)

Relative mode takes and optionally transforms relative or absolute source control values. *Relative* means that the current target value is relevant and the change of the target value is calculated in terms of increments or decrements. This is most often used with control elements that emit relative control values, such as rotary encoders. In fact, this is the only mode that works with rotary encoders. Therefore, if *Auto-correct settings* is enabled, this mode will automatically be activated if the source is known to emit relative values (increments/decrements).

You read correctly, ReaLarn's relative mode can also deal with incoming absolute control values - provided they are emitted by button-like control elements. Let's assume you use the *Note velocity* source together with relative 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 it velocity-sensitive). If you want the target value to decrease, you need to check the *Reverse* checkbox.

- **Step size Min/Max:** 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.
    - *Step size Max* is used to limit the effect of encoder acceleration (in case of rotary encoders which support this) or changes in velocity (in case of velocity-sensitive control elements). If you set this to the same value like *Step size Min*, encoder acceleration or changes in velocity will have absolutely no effect on the incrementation/decrementation amount. If you set it to 100%, the effect is maximized.
- **Speed Min/Max:** When you choose a discrete target, the *Step size* label will change into *Speed*. *Discrete* means there's a concrete number of possible values - it's the opposite of *continuous*. If a target is discrete, it cannot have arbitrarily small step sizes. It rather has one predefined atomic step size which never should be deceeded. So allowing arbitrary step size adjustment wouldn't make sense. That's why *Speed* instead allows you to *multiply* (positive numbers) or *"divide"* (negative numbers) value increments with a factor. Negative numbers are most useful for rotary encoders because they will essentially lower their sensitivity. Example:
    - Let's assume you selected the discrete target *FX preset*, 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.
    - 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.

- **Reverse:** If checked, this inverses the direction of the change. Increments will be translated to decrements and vice versa.
- **Rotate:** 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.
- **Feedback transformation (EEL):** This has the same effect like in absolute mode (feedback is always absolute, never relative).

3.2.5.3 Toggle (for buttons only)

Toggle mode is a very simple mode that takes and optionally transforms absolute source control values. It's used to toggle a target between *on* and *off* states. Only makes sense for button-like control elements.

- **Length Min/Max:** Just like in *Absolute mode*, this decides how long a button needs to be pressed to have an effect. Unlike in *Absolute mode*, here the achieved effect is not *triggering* but *toggling* the target.
- **Feedback transformation (EEL):** This has the same effect like in absolute mode.

Important: Sometimes the controller itself provides a toggle mode for buttons. Don't use this! 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.

4 Automation and rendering

Similarly to control surfaces, ReaLearn is meant to be used for controlling targets "live". If you want to *persist* the resulting target value changes, you can do so by writing automation. Just like with any other automation, it will be included when you render your project.

It *is* possible to feed ReaLearn with track MIDI items instead of live MIDI data. This also results in a kind of automation. **But be aware: This kind of "automation" will only be rendered in REAPER's "Online Render" mode. It will be ignored when using one of the offline modes!**