

Audiate User Manual

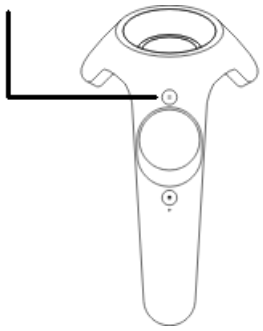
This document is automatically generated from the in-application help pages. Aside from this title page both are identical.

Copyright 2018, 2019, & 2020 Fraser K Consulting Ltd. All rights reserved.

Introduction

Audiate translates VR position & rotation along with button inputs into MIDI messages. These can then be used to control MIDI capable software or hardware.

Hold controller button for 1-sec to recentre controller.



Note: SteamVR now puts devices to sleep, even when plugged into the mains. It's recommended that you disable this for the controllers, via: *SteamVR Settings / "Startup / Shutdown"*. It should not be necessary to change the headset setting.

Information on interfacing with various DAWs can be found in the menu to the left. Some require some brief configuration before they'll work.

A “virtual MIDI port” is required to communicate with software applications, information on this and several free implementations can be found online.

Copyright 2018, 2019, & 2020

Inputs

Position

Each controller provides three positional axis inputs:

- left-right
- down-up
- backwards-forwards

Position varies by: +/- 10 cm from centre (configuable in options)

Rotation

Each controller provides three rotational axis inputs

- pitch
- yaw
- roll

Rotation varies by: +/- 90° from centre

Velocity

All position and rotation inputs also have a corresponding velocity value. Smoothing is enabled by default on these inputs as driving them to large values for any more than a brief moment is difficult. The max reference velocity and smoothing can be set/disabled in the configuration files.

Button Axis

Different devices have different buttons available, with some ranging from "none" in tracking devices to "many" for those with triggers, buttons and trackpads.

Some buttons provide for proportional input while others are purely digital on/off switches. Many also have separate touch / press sensors, each can also be configured as an input if desired.

All available OpenVR inputs are supported via configuration files, see the section "Input Device Configuration" for more info.

The default inputs on the HTC Vive are as follows:

- left-right proportional input from trackpad *when touched*
- down-up proportional input from trackpad *when touched*
- left-right proportional input from trackpad *when clicked*
- down-up proportional input from trackpad *when clicked*
- trigger proportional input
- grip button digital input

Note that the trackpad is available separately when touched and clicked. This form of "constraint" can be added to any button, for example you could add a new input chain that only outputs the "position left-right" input value if a particular button is pressed.

Please see the Input Device Configuration section for more details on advanced configuration options.

Supported Devices

Copyright 2018, 2019, & 2020

All OpenVR devices should be supported via the Generic Controller and Generic Tracker configurations, however this may not provide all available inputs on each device.

For best results a custom profile should be created, one that maps every input and provides distinct names for things like left & right hands. More information on this can be found on the Steam forum on how to do it yourself or request assistance to have it done for you.

Profiles are pre-installed for the following devices:

- Valve Index aka "Knuckles" (see note)
- HTC Vive Controller
- HTC Vive Tracker
- Oculus Rift Controller
- Windows MR Controller

Note on Valve Index Controllers:

Index controllers are provided via the OpenVR 1.0 API which only exposes some of their sensor values. Functionality will be the same as is seen in other "Legacy Mode" applications.

Supported functionality includes the base OpenVR position/rotation/trigger/grip values seen on almost all devices. It does not have the fine-grained per-finger knuckles data. It does however include a few extra sensors over the base legacy inputs such as binary touch sense on trigger & grip.

The additional Index features are only available via a new "SteamVR Input" developed alongside the controllers, but not through the original OpenVR API that Audiate uses.

Support for this is planned in Audiate 2.x which will be a free upgrade for all 1.x users.

Copyright 2018, 2019, & 2020

Mapping Inputs

Input Devices

The main panel lists every available device, listed in they order the were found by the application.

A "recentre" button has been provided for each device, this duplicates the function of the reset button on the device itself. Generally speaking it's easier to use the on-device button, the UI equivalent has been provided to support devices that have no buttons.

Input Chains

Each device has a list of available inputs under it. These are customisable via configuration files, this can be done to support new types of OpenVR devices or to further customise one of the pre-configured devices. The specifics of this are covered in a dedicated section later in this manual.

Each input is fed into a "chain" which controls what (if anything) is done with it. These flow from the left to the right, much like a mixer flows from top to bottom.

A graph is shown next to the name of each chain showing the present value along with the recent history of the value. This functions much like a stock ticker graph, with time along the horizontal axis and value on the vertical. There is a convention in the input naming where the lower value is listed first, for example "left-right" implies that left is lowest and right is largest. The graph can help visualise this.

Next to the graph is the status display. If this is blank then it means that the chain is active, the purpose of this display being to help identify anything that may be misconfigured. Any errors are shown in a red typeface. If an error is associated with a particular field then that field will also be highlighted with a red border.

Input Chain Controls

The Solo button functions exactly like that on a mixer, with the exception that clicking Solo on a second device disables Solo on the previous one. This is provided to assist mapping in software that learn mappings through listening for MIDI data. Audiate may be broadcasting many signals at once & the Solo button allows the others to be temporarily disabled during mapping. It can of course also be used for Solo'ing one input for any other reason should one want to do so.

The Solo button doubles as an indicator, glowing when active. When active the Solo button on every other chain will be highlighted with a red border and the chain status will indicate that Solo mode is active.

The "Enable" button allows the user to enable & disable each chain. As with the Solo button this glows to indicate that it is active.

Note that Solo does not override the Enable button for chain. If a chain is not enabled then setting Solo will not cause it to be sent.

Input Chain Device Configuration

Each chain contains a drop down menu list each available device. There is no requirement that a chain can only go to one device, every input on each controller could be configured to go to a different device.

Output is modular, with MIDI being the only implemented protocol at present. Future versions may provide alternate types of data such as bespoke MIDI sysex commands for emulating a specific controller or perhaps other protocols entirely.

MIDI Output Configuration

Audiate can send various different types of MIDI message though for the most part the "Control Change" messages will be used almost exclusively. Program Change messages are expected to be of little use here as they don't tend to be used in conjunction with control surface data. Some users may find the Pitch Bend message type useful. Aftertouch support is provided though few MIDI devices are capable of doing anything

Copyright 2018, 2019, & 2020

useful with it.

Once a MIDI output type has been selected the application will display any relevant configuration for that message type. Most MIDI messages require a channel number as a bare minimum.

Control Change messages are far more configurable and are covered in a dedicated section of this manual. If you are unfamiliar with these it is strongly recommended that you first read the MIDI Primer section of the manual before proceeding. If you know the basics you can skip directly to the "MIDI Control Change Options" section to learn about the various options available.

Copyright 2018, 2019, & 2020

Patches

Patches

A configuration can be stored as a Patch and retrieved later. To do so first select the Patch entry by clicking on it then click on the save button.

The selected Patch can be loaded by clicking the load button or by double-clicking on the entry.

To rename a Patch first select it then click once on the name.

The files saved by the application to store patches are simple plain-text properties files stored in the user profile "Application Data" directory. These can be edited by hand if desired.

Banks

Each Bank contains up to 128 Patches. There are two kinds of Banks:

- User
- Library

There is only one User Bank at present, this may expand in future.

Library Banks are a special kind of Bank which cannot be edited. These provide starting points & working examples that are intended to be tweaked further for use. Changes can be saved as User Patches.

Auto-Collapse

The user interface can automatically expand or collapse devices & groups when loading patches. This feature can be enabled or disabled for either type in the Settings/Misc page.

This is disabled at the device level by default as it is best used when there is a large number of input devices.

Copyright 2018, 2019, & 2020

MIDI Primer

Introduction

This section serves as an introduction to MIDI. If you are already familar with things like Control Codes then you can skip over this entire section.

MIDI consists of musical note data and a variety of control messages. Well known control messages include “pitch bend” and “modulation”, sometimes included as hardware controls on some keyboards.

Instruments such as keyboards tend to focus on note data. "Control Surfaces" focus more and typically solely on control messages. This application uses VR controllers as a Control Surface.

Most MIDI capable devices support some control values while others allow you to customise all aspects of their sound abilities. Software-based MIDI devices such as VSTs typically set some set up by default while allowing the user to map custom ones themselves.

The Basics

Each MIDI “port” consists of 16 channels, numbering 1-16. A small number of devices list these as 0-15 & it’s not unusual to have to set links up where it’s listed as “0” on one side and “1” on the other. This app uses 1-16 throughout it's UI. Saved patch files use 0-15 as is common for saved MIDI data though most users will never have a need to interact with these.

Systems can have multiple ports to expand on the 16-channel limit. Adding dedicated ports for each instrument makes things a lot easier to manage as you can glance that any control and see what it's doing by name instead of having to remember what "channel 4" is set to.

MIDI also has some limitations in its bandwidth, being a fairly old serial protocol. This is a concern when you are using a lot of control data messages along with a large number of musical notes on the same port, problems include audibly delayed notes due to message queuing. While you need to have considerable volumes of data to hit this problem it is generally preferable to have each instrument on its own port if possible, making use of as many ports as are required.

Two software devices can be made to talk to each other using a Virtual MIDI port. There are numerous free & commercial tools available online to facilitate this.

Control Changes

Most controls are accessed via “Control Change” messages. Common examples include ADSR settings, pan/volume, input sound form, vibrato as well as various effect filters such as reverbs.

Hardware devices tend to have a section in the back of their manual or a separate “MIDI Implementation Document” describing those that are available. Software DAWs such as Pro-Tools and Ableton allow you to map many aspects of their UI to midi controls, offering the greatest scope for customisation.

“Control Change” messages consist of the "channel" they are on, a “number” defining which control, and a “value” for that control. Both "number" and "value" range from 0-127 although the “number” values from 121-127 inclusive are special commands known as “Channel Mode” messages. These perform functions such as turning off all active sounds & should be avoided unless you specifically want to make use of the function.

While there are many well-known messages defined in the official MIDI specification manufacturers are free to use them for any purpose. Typically they’ll try to conform to the standard numbers so-far as possible.

Copyright 2018, 2019, & 2020

MIDI Implementation

Output

- Pitch bend messages
- Control change messages (any ID)
- Program Change messages
- Aftertouch

Input

- Program Change messages

Output Details

Each available input is presented as a row in the table. A graph shows the present input value from 0-1. As an example the left-right input will read 0 at the left-most position and 1 at the right-most.

If a MIDI channel is selected along with an output type then this will result in a stream of data for that value. The value is scaled up to the range supported by the control, in most cases 0-127.

Some output modes allow restricting the output value to a min/max range. If min is set to 50 and max to 100 then the left-most position would output 50 and right-most 100, with the positions in-between scaling linearly between the values.

Input Details

Patches can be loaded automatically using Program Change messages. These can be received from any MIDI input device, for example a footswitch or a sequencer.

This feature can be enabled in the Settings "MIDI Input" page by selecting the desired input device. You can restrict this to a single channel or respond to any channel that device.

Copyright 2018, 2019, & 2020

MIDI Control Change Options

Various options are available to configure how Control Change data is sent from the application. This helps interface with different applications, some of which are limited in their MIDI abilities.

It's recommended that "Absolute" mode is used where possible. Controls that are not compatible with "Absolute" will jump around wildly, often between the lowest, mid and highest settings. When this happens the other options such as "Relative (window)" should be considered.

This is caused due to the control expecting "relative" input; this section details how to solve it.

There is also a short section in the "Appendix - Endless Knobs" section below explaining the root cause of the problem & where it originates.

Absolute

This mode is a simple scaling of the value from 0-127, based on the input value 0-1.

The range can be reduced from the full 0-127 range, for example setting it to 0-63 would scale the input across only the lower half of the range.

The range can also be reversed by setting the first number as higher than the second. For example a range of 127-0 for "left-right" would increase the value when moved to the left.

As stated above this mode is preferred where possible but is incompatible certain controls.

Relative (window)

This mode emulates "Absolute" mode as far as is possible using relative control messages. There are some drawbacks to this.

The main drawback of this mode is that the value in the DAW might not match the one that's desired if something external were to change the control (like track automation). Absolute mode would simply overwrite the value and work normally but as Relative mode involves moving up and down we lose track.

This can be easily solved by briefly moving the input across the full range at any time. It's required that the minimum and maximum values are touched just once to allow the control to re-sync with the DAW.

To explain further, imagine sending text messages asking a friend in an elevator to go up and down 1,2, or 3 floors at a time. If you know where they were when you started then you'd know where they are by keeping track. If another friend sends them a message telling them to go up 5 floors then your own idea of their position is now wrong. But if you send them a message saying "go to the bottom" then you once again know exactly where they are.

A lesser downside of this approach is that the output range cannot be reduced. It's necessary to be able to go from min to max values to sync both sides.

The "tick" option is used to set how many ticks there are across the whole range. This will likely be a power of two in most cases such as 128, 256, or 512. The default value of 256 is suitable for most applications.

Relative (brute)

This is a somewhat dirty hack that *may* work in some situations to emulate absolute input for a relative control. Every time a message is sent the control is first driven to it's absolute minimum value then it is brought back up to the desired value.

This has the advantage of always being in sync on both sides, regardless of external changes to the control. There is no need to re-sync as there is with "Relative (window)" mode.

The downside is that the driving of the control down to minimum requires sending 4 or 5 control messages then as many as 3 messages to drive it back up. This might not be welcome with some controls, resulting in noticeable glitches.

Copyright 2018, 2019, & 2020

Relative (bias)

This mode takes a completely different approach & offers a new form of input. It operates much like a joystick on a gamepad. If held in or near the centre position (the deadzone) then there will be no change to the value. The value can be increased or decreased by moving the input out of the deadzone. The farther it is away from centre the faster the value will fall or rise. Once returned to the deadzone the last value will hold until further changes are made.

The main advantage of this is that relative controls can be modified without worrying about where they are.

The size of the deadzone can be configured with the slider. This is as a percentage of the full range of the control. If this is set to half-way then half of the entire input range will be in the deadzone.

Appendix - Endless Knobs

aka "Why do we even need the above options?"

Full-featured Control Surfaces have motorised controls to physically move a knob or fader to a value, allowing for record/playback of the setting. These are expensive, big, and heavy devices whose most commonly used feature seems to be the demo mode that shows off these abilities!

The consumer-grade Control Surfaces available on the market provide a compromise with "endless knobs". Instead of driving a traditional knob with fixed start/end points they have a knob that can be spun in one direction continuously. As the user turns the knob "ticks" are sent showing the progression. Turning it faster results in more ticks. The data sent is *relative* to the current value and not an *absolute* value.

These knobs are significantly cheaper to manufacture than the motorised devices but as might be expected they don't move themselves. In this situation having fixed start/end points on the knob would be a huge problem because the physical device might be set at minimum while the software thinks it's at maximum. The control would then be stuck until the user manually turned it all the way up to max then back down again.

Moving a fader value like this to sync is sometimes known as "pick-up". Some DAWs have options to configure how these situations are handled, for example Ableton's "Takeover Mode" setting relates to how such jumps should be handled on absolute controls like faders. Ableton itself is fine with Absolute mode throughout though it can work with relative mode if desired (not recommended).

If a DAW control *only* accepts relative input then it is necessary to emulate it. There are different ways to approach this, as has been described in this section of the manual.

Copyright 2018, 2019, & 2020

MIDI Control Change Names

Clicking on the arrow in the Control Change field presents a dialog box listing various control changes by name. Various types are provided with the application & the user can also easily import or create their own as described towards the end of this section.

Standard

This grouping lists all of the CC numbers according to the MIDI standard.

Hypercontrol

Hypercontrol is used by some software to control plugins that do not have native support.

The "Pan knob" ones can be mapped to rotary values and almost always use a Relative mode. "Fader" ones work with linear fader-style controls and tend to use Absolute mode.

Unfortunately the standard only offers eight of each control.

Hypercontrol (subset)

This grouping simply contains the most commonly used Hypercontrol controls i.e. pans & faders.

ProTools - Vacuum

Vacuum is one of the ProTools plugins that provides full MIDI support. These mappings are the default ones available in the module. Sorting the grouping by name is recommended as the numbers are non-contiguous.

Note that not all controls come with pre-mapped values. You may want to map these yourself if they are needed. if you do so you could make a copy of this CC name file and add yours to it.

Custom Groups

A user can add in as many of their own custom names as desired. These are stored in:

%APPDATA%/Audiate/MidiControlCodes/

Two different file formats are supported. The first is industry standard midnam files which can be found online. The other is a simple csv file which some users may prefer if they are creating their own & they are not familiar with XML.

Both types support using the file name to control the order they are displayed in the UI. To do this simply start the name with two digits and a dash as follows:

01-First.csv
02-Second.midnam
03-Third.csv
04-Fourth.csv

When you do this Audiate will know to hide the number and dash in the UI. Files without a sort order will appear alphabetically after the ones that do.

The order of the name entries within each file does not matter in either file format, the UI allows you to sort by name or number.

Copyright 2018, 2019, & 2020

Custom Names - CSV

Each CSV file contains a single grouping. The data is csv format which can be easily edited in most spreadsheet applications like Excel or Open Office, or a simple text editor like Notepad.

They look like:

```
Number,Control Function
0,Bank Select
1,Modulation Wheel or Lever
```

Only columns 0, 1 (A & B) are used at present. The first row is always the title row which is ignored when parsing.

Note that some editors like Excel lock the file while it's open, preventing other apps like Auditate from reading them. In these cases you will need to close the editor before running the app. Simple text editors tend not to require this.

Custom Names - Midnam

A file with a *.midnam extension is an xml document that lists patch names for instruments and/or control numbers for the controllers.

Midnam files for various devices can be found online. Searching for "midnam " tends to find them if they exist.

Note that not all midnam files contain control numbers, most only contain the patch names. The ones that contain control numbers have a section near the end under a set of ControlNameList/Control elements.

The files are NOT validated against the schema on loading, this has been disabled to avoid the need to resolve external addresses on the internet as some of the older ones no longer exist. However it's recommended that if you use an XML-validating editor that you run the validation prior to saving. If you do not know what this means then don't worry about it, validation is just a way to verify a file, saving you time if there is an error.

Copyright 2018, 2019, & 2020

Hints and Tips

General:

- Use a different virtual MIDI port for each instrument/plugin, with its name set to something meaningful.
- Limit the range of the output to the useful range of the control.
- The recentre button can also be used to identify controllers, provided haptics are enabled or the device has an LED.

Different kinds of input can be particularly useful for different kinds of control:

- The trigger is ideal for controls that will set a set value most of the time and be tweaked occasionally then returned to the original value.
- The pad is useful for values you want to set once and then leave on that value for a while.
- Rotation inputs work best for things that are generally held about the middle the moved up and down for effect.
- Rotation inputs can usually be swept across their range more quickly than position as it's just a flick of the wrist.
- Position inputs are ideal for things that rise and fall slowly with a more intuative feel to them.

Different ways of setting the centre can be useful:

- Each controller has it's own centre position which doesn't need to be anywhere near the other one. Another person could use it for example.
- Rotation doesn't have to be set and level, it's taken relative to where the controller is pointing when you press the re-calibrate button. As with the position inputs each controller is independent of the other.

Copyright 2018, 2019, & 2020

Input Device Configuration

Update note:

This subsystem will be changing significantly in Audiate 2.0.

Audiate 1.x will continue to be maintained for bug-fixes, this will be made available via a special 1.x branch on Steam.

Background: it had been intended to provide a UI within Audiate to configure all of the settings below, this page was intended as a temporary stop-gap for users that did not mind a more hands-on approach. The plan was to allow users to share configurations via the Steam Community Workshop, users were not expected to edit the files by hand. The data format is designed to be easy to synchronise & merge in software, being easy to read for humans was not a primary goal.

As work began on this UI it was announced that OpenVR would get it's own mapping system that did all of the above. It makes far more sense for Audiate to use the same system that users already know and understand from other software.

The unfortunate downside was that the new OpenVR Input system would not be stable until Summer 2019, coinciding with the Valve Index release which it is closely linked to. This means that the hands-on system described below has lived on for far longer than had been originally intended.

Overview

Audiate supports OpenVR controllers & trackers by using a configuration-driven approach. The software knows nothing about Oculus or Vive VR hardware and merely interfaces with their devices via the OpenVR sub-system. It is down to configuration files that define what each specific device can do, ultimately resulting in a list of input chains presented to the user.

This allows the user to completely change how the application presents all of the inputs, including adding support for new OpenVR-compatible devices on the market in years to come. Some devices already have setups provided, others will be added as information becomes available on them.

A user could:

- list the same input multiple times to send it off to several places
- remove or customise a default
- constrain an input to work only when a button is pressed
- add support for a new kind of device
- make use of tracker pogo pins to add new kinds of input

Help is available on the forums should you need any assistance. If you'd like support added for a popular device that is not yet provided by default then please feel free to ask for it to be done. If you do it yourself then please consider sharing it on the forums for the benefit of others.

The log file (see below) contains a couple of lines of information that would be extremely useful alongside any input device discussion on the forums. Look for lines containing "supportedButtons", for example:

> Device info [modelName=Vive Controller MV, renderModelName=vr_controller_vive_1_5, supportedButtons=[0, 1, 2, 31, 32, 33], axisTypes=[TrackPad, Trigger, , ,]]

Log File

Before editing anything you should probably first familiarise yourself with the application log file. Any errors made in configuration will result in information being logged to that file. It can be found at:

> %LOCALAPPDATA%\Audiate\log\messages.log

Useful information is also logged here on application startup about the available input and output devices.

There are applications available online such as "LogExpert" that'll show a log file in real-time, this approach is highly recommended.

Copyright 2018, 2019, & 2020

Configuration files

Users can customise all input device settings via a config file in their user directory, located at:

%APPDATA%/Audiate/input_devices.properties

These settings override those found in a file of the same name inside the application installation directory, in the "Config" directory. Taking a look at that would be a good place to start.

While you can edit the file inside the installation this is not recommended as changes will be lost on application upgrades. Any and all properties can be overridden in the user-level file, there should be no need to edit the base file.

A quick note on properties files

Property-style files were chosen for the application because it makes them easy to override at various levels & they are more user-friendly for less experienced users.

A property file is a simple plain text file containing a list of "key-value pairs", one on each line. Each has a key (the name) and a value. They look like:

the.key.to.the.value=the value of the value

The order of each entry in the file does not matter in any way.

Lines beginning with a # symbol are comments that are ignored by the application:

```
# this describes the thing
the.thing.being.described=value
```

To remove a property via an override simply set it to no value:

property.that.I.do.not.want=

Note that if you want to remove something there may be a specific "enabled" property that provides a more convenient method than setting a whole bunch of values to nothing.

The Basics

The mapping format has been designed to minimize repetition. This is done by separating the concepts of a device "model" from the actual defined inputs listed in a "profile". For example most devices share the "OpenVR-Tracking" profile which provides the position and rotation values.

Various profiles exist such as "OpenVR-TrackpadOnAxis0" or "OpenVR-TriggerOnAxis1x" which can then be used by many different "Model"s.

In addition to reducing repetition this also provides for some level of cross-compatibility between similar but different devices. Each profile has it's own set of property names used when saving a patch to disk and devices sharing that profile then know what to do with them. This can also work across profiles if desired, for example you might want to add "OpenVR-TriggerOnAxis*2*x" for some new device and re-use "Trigger" as the property name for compatibility. This would not be recommended on a device that provides a trigger on both axis of course! In such cases a new "TriggerSecondary" property would be logical.

This can also be used with tracking device pogo pins, allowing you to add custom buttons if need be. If you do something interesting with this then please tell us about it on the forum.

Model

The Model properties bind specific OpenVR model names to their profiles. Each "Model" in Audiate can actually refer to several different types of

Copyright 2018, 2019, & 2020

device where they are logically the same. This makes use of the "render model" property in OpenVR which at the time of writing represents the favoured way to determine what devices are connected.

Of most importance is the model name as seen in every property relating to it:

```
input.model.openvr.ViveController.blah.blah=123
input.model.openvr.ViveController.more.blah=456
```

Here it is "ViveController" in these imaginary properties. This value can be anything so long as it's unique. The value should be something describes the model in some way but note that it's never seen outside of this file.

Each model you define requires the following three properties as a minimum:

```
input.model.openvr.ViveController.enabled=true
input.model.openvr.ViveController.label=Vive Controller
input.model.openvr.ViveController.prop=VRController
```

"enabled" - This is provided should you want to quickly disable it without having to comment out the rest. Valid values are "true" and "false" (without quotes)

"label" - Informs the application what label to use in the UI, can be literally anything you want

"prop" - Used to set the name of the device to save in patches and when loading libraries

Most of these are self-explanatory. The "prop" setting does need some thought if you want to share patches with other people. This is the top-most name defining where different patch values are stored so choosing a different name means complete incompatibility between patches. This may be desired in some cases where there is little commonality between devices.

Model Variants

A Model definition doesn't do anything until you bind it to a "variant":

```
input.model.openvr.ViveController.variant.0.id=vr_controller_vive_1_5
input.model.openvr.ViveController.variant.0.label=Vive Controller
input.model.openvr.ViveController.variant.0.centre.position={0,0,-100}

input.model.openvr.ViveController.variant.1.idKey=Prop_ManufacturerName_String
input.model.openvr.ViveController.variant.1.id=vr_mind_reader
input.model.openvr.ViveController.variant.1.label=Future Tech
input.model.openvr.ViveController.variant.1.centre.position={0,0,0}
```

This defines the various device names used to recognise the device from what is available on the system.

Note how there is a number following "variant". This does not have to be a number, it merely needs to be unique for each line. As you might think this also means that gaps do not matter, you could number them 1, 2, 3, 5, 8 if you wanted. Anything so long as it's unique.

The properties here are:

"idKey" - (optional) controls which OpenVR property key is used to identify the device, see below.

"id" - corresponds to the unique value used to identify a device, using whatever key idKey points to.

"label" - used in the UI wherever individual device types are listed.

"centre" - sets the default device centre to set in the UI Configuration page, defaults to 0,0,0 when omitted.

If "idKey" is omitted then it defaults to "Prop_RenderModelName_String" which can be seen in the logs.

A user might want to change this to:

- support a device that reports a "Prop_RenderModelName_String" which changes or is missing/blank.

- identify a device by serial number so it can be given a unique configuration different from others.

Copyright 2018, 2019, & 2020

Any ETrackedDeviceProperty is valid, the most obvious candidates are:

- Prop_ModelNumber_String

- Prop_SerialNumber_String

- Prop_ManufacturerName_String

- Prop_RenderModelName_String

These properties are logged for each device on start up & their values can be cut & pasted directly from the logs.

Model Profiles

The final part of the Model is where you bind it to the available profiles:

```
input.model.openvr.ViveController.profile.0=OpenVR-Tracking
input.model.openvr.ViveController.profile.1=OpenVR-TrackpadOnAxis0
input.model.openvr.ViveController.profile.2=OpenVR-TriggerOnAxis1x
input.model.openvr.ViveController.profile.3=OpenVR-GripOnButton2
```

This is fairly straightforward, listing each of the profiles, one per line. As with the alias the number following "profile" does not need to be a number, merely something unique. Each entry pulls in all of the available inputs defined in that profile. These may or may not be arranged in sub-groups depending on the profile.

Profiles

A Profile lists one or more inputs that will be seen by the user as an input chain. Where multiple inputs are provided there will be some commonality between them, for example a trackpad can produce up to four different inputs using both axis and both touch and click sense.

For the purposes of this example we'll use the pre-configured "OpenVR-TriggerOnAxis1x" Profile. This name will be seen in every property relating to it.

To define a Profile you must first tell it what input module to use via the "provider" key:

```
input.profile.OpenVR-TriggerOnAxis1x.provider=openvr
```

Next you need to define a group for the inputs to appear in. Each Profile can have multiple groups, for example the "OpenVR-Tracking" one has a position and rotation group, each with three inputs.

```
input.profile.OpenVR-TriggerOnAxis1x.groups.0.label=Trigger
```

As with previous properties the last-most number in the name does not need to be a number, it only needs to be unique. Groups only have one property, "label", which sets the name seen for the group in the UI.

The remainder of the section deals with setting up each input. Note that you can, if desired, set up the same input parameters multiple times so that the same source value appears as multiple chains. This is useful in conjunction with constraints (discussed later in this manual).

Profile Inputs

Each input is defined under the "chains" property as follows:

```
input.profile.OpenVR-TriggerOnAxis1x.chains.Trigger.label=trigger
input.profile.OpenVR-TriggerOnAxis1x.chains.Trigger.group=0
```

The "Trigger" part of the name here defines the name of the property used to save patch files. If you want compatibility between your new device and other pre-existing ones then you should choose the same names as used there.

 Copyright 2018, 2019, & 2020

One thing to note is that this does need to be unique per device. If two separate Profiles define "Trigger" and both are referenced on the same Model then there will be confusion when it comes to saving and loading patches.

"label" - defines the label seen in the UI which can be anything you want.

"group" - the group we referenced when creating the Profile. Each appears under it's own sub-heading in the UI.

Profile Input Source

```
input.profile.OpenVR-TriggerOnAxis1x.chains.Trigger.source.type=axis2d
input.profile.OpenVR-TriggerOnAxis1x.chains.Trigger.source.id=1
input.profile.OpenVR-TriggerOnAxis1x.chains.Trigger.source.param=x
```

The "source" values are a little more complex and some basic understanding of OpenVR goes a long way. The free tool "OpenVR-Inspector" can be invaluable in determining how a devices buttons are exposed in the API.

The API provides a wealth of info on each device. Most devices provide tracking information as a base. Additional inputs are then provided as a series of "axis" and state values which you can reference in Audiate.

Firstmost, each source has a "type". These are:

axis3d - a value that comes from the tracking system

axis2d - a value that comes from a proportional button input or trackpad

binary - an on/off value from a button or touch sense

Each source then requires two properties, "id" and "param" telling it which one of that type to use. These have different valid values for each "type" so they'll be described separately.

For type "axis3d", the "id" property must be either "position" or "rotation". Each of these has three values available for the "param" property: "x", "y", and "z". As you might expect these x, y and z values refer to the 3-dimensional values from position and rotation.

For type "axis2d", the ID property refers to the numerical index of the axis. These are where inputs like triggers, joysticks and trackpads are found, as well as variable grip inputs.

OpenVR currently supports five of these axis, numbered 0 through 4. Determining which is which on a completely new controller is where "OpenVR-Inspector" can be useful. It displays the values for all five when a device is selected, allowing you to interact with the device to see which each control physical control is exposed.

The "axis2d" type has only two valid values for "param", specifically "x" and "y". Some inputs like trackpads use both, providing a two-dimensional value that can be used to provide two different input chains. Other inputs like triggers might only use the "x" one, leaving "y" always set at 0.

Note: if configuring a trackpad-like touch device you should also see the "Constraints" section below.

For type "binary" there 64 theoretically possible values of "id", numbered 0 to 63 however SteamVR reserves "0" for the Home/Power button & Audiate reserves "1" for the "recentre" function so neither of these can be used here. The value of "param" can either be "pressed" or "touched". Some devices do not provide touch sense on all buttons & on those the "touched" one produces the same result as "pressed".

The list of available buttons for each device is displayed in the log file, as "supportedButtons".

The following IDs are common:

0 - System (unassignable, used by SteamVR)
1 - Menu (unassignable, used by Audiate)
32 - Axis 0
33 - Axis 1
34 - Axis 2
35 - Axis 3
36 - Axis 4

The buttons associated with "axis" are usually related to the axis, for example if the axis was a trackpad or joystick then the button will mapped to

 Copyright 2018, 2019, & 2020

the user clicking it. This is not an absolute rule & devices could potentially deviate from it.

There are a series of Appendixes at the end of this section listing known devices and their buttons.

Information on button IDs can also usually be found online as it's essential to anyone writing software for that device. If you are stuck then making a post on the Audiate Steam forums might help. If in doubt you could always assign a whole bunch of them using copy/paste covering the ones listed in the log then try them out in the app to see if any react to inputs you then provide. Inelegant but effective.

Note: it's possible that some devices may not be fully accurate in their listing. The HTC Vive controller lists "31" for example which doesn't seem to do anything.

Constraints

All inputs regardless of source type can also be "constrained" by a button. When configured the output will only be read when that button is also touched or pressed.

This feature is essential to provide for trackpad input as these pads tend to output their mid-point value when not touched. It isn't just limited to this though, it can be used to constrain any input to any button.

In simple terms this means you could have a specific input or perhaps several of them that only function when e.g. the grip is pressed. When the button is released the value is held at it's last value until when it's next pressed.

More elaborately you could set up two or more buttons to do this with a number of inputs, switching between which ones are active depending on which button is pressed. If do consider doing something this then please start a thread on the forums stating what the aim is. Depending on the need it may make sense to add new application features to make it easier to set up & even more flexible.

Constraints are easy to set up, simply add a "constraint" property to the chain in the Profile:

```
input.profile.OpenVR-TrackpadOnAxis0.chains.PadYClick.constraint.pressButton=32
```

This example sets a constraint on the "OpenVR-TrackpadOnAxis0" profile, on the "PadYClick" chain, telling it to only function when button 32 is pressed. This is used on the HTC Vive to bind the trackpad input to it's associated button.

The text in the name after "constraint" can be either "pressButton" or "touchButton" to bind this constraint to either interaction type. As noted above not all devices provide touch-sense on all controls, some only set this when it's fully pressed.

The number corresponds to the button number as described above in the binary source section.

Appendix - HTC Vive Controller

Axis2d:

- 0 - Trackpad on x & y, see also button 32
- 1 - Trigger on x only, see also with button 33

Buttons:

- 0 - System (unassignable, used by SteamVR)
- 1 - Menu (unassignable, used by Audiate)
- 2 - Grip, press only
- 32 - Trackpad touch and press
- 33 - Trigger touch on partial pull, press on full pull

Copyright 2018, 2019, & 2020

Ableton

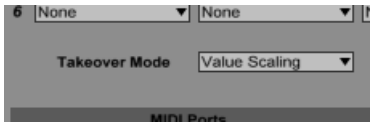
Ableton can be communicated with via a Virtual MIDI port.

Important one-time setup

Ableton ignores all control inputs on MIDI ports by default. It is essential that all virtual MIDI ports have been configured in the Ableton settings to accept “Remote” messages. This setting can be found under Options/Preferences/MIDI (as shown).



Above this can be found the “Takeover Mode” option, this works best when set to “Value Scaling”. This can prevent issues with controls that stop responding:



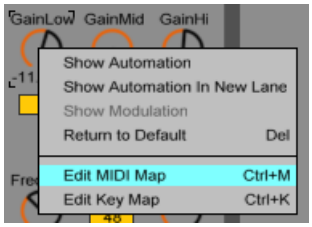
Mapping

Controls can be mapped as follows:

Enable “Solo” for the desired input in Audiate:



Right-click on the control in Ableton and select “Edit MIDI Map”:



Verify “mapping” drop down in the very bottom-left of the Ableton window is set to “Absolute”:



Verify that the correct value is shown as channel/number on the control.

Additional items can be mapped at this point by clicking them in Ableton then activating “Solo” in Audiate for the next input. To end the mapping mode in Ableton simply right-click on the control again and turn off the Edit option, or click the “MIDI” button in the top right of Ableton.

Copyright 2018, 2019, & 2020

Saving configurations

Mapped controls can be saved in Ableton and re-used by dragging the plugin back into the library. This will persist all of it's settings including MIDI mapping.

An alternate approach is to incorporate it into a template session that can be used as a starting point for new projects.

Copyright 2018, 2019, & 2020

Pro Tools

Pro-Tools can be interfaced with using a Virtual MIDI port.

There are two ways to map controls depending on the level of support provided by the desired control. If the control supports direct MIDI mapping then it's recommend that it is used.

MIDI Mapping

Some controls offer direct MIDI mapping where they learn a Control Code. This is the most flexible method as it allows any CC number to be used.

These controls generally work perfectly with the Absolute control mode.

Controls that offer this usually show an option similar to the following when right-clicking on them:



Some plugins have a menu or config page where mapping is configured.

In this example the CC number of 14 has already been assigned as a default value for the control. You can use this as-is or change it to something else if you prefer.

To change the mapping first activate Solo on the desired input in Audiate:



Then click on the "Learn MIDI CC" option on the control. Don't forget to deactivate Solo once it's done or alternatively you can Solo the next input to learn another control.

Hypercontrol

NOTE: This must be enabled in the ProTools options for it to work. See below.

Many controls do not offer direct mapping and must be controlled via different means. One method is to use the same CC numbers that are used by a standard set of hardware controllers on the market.

There are some downsides to this approach:

- As with a hardware controllers the plugin may need to be in focus for the controls to work
- Pro Tools can lose focus on it's own sometimes
- Sending controls to the wrong plugin through changing focus can result in changing parameters you did not want to change
- There are only a limited number of values available

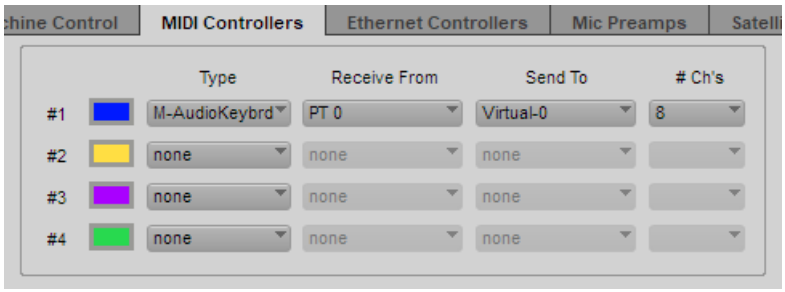
These issues are common to all Pro Tools control surfaces due to the way they work.

Pro Tools bug note: If a project in Pro Tools is saved with the plugin open then that plugin will NOT work properly when the project has been reloaded. The workaround is to close the plugin window in Pro Tools then re-open it. This works most of the time but sometimes the control in

Copyright 2018, 2019, & 2020

question may need to be re-mapped.

To enable this option in ProTools you need to configure a MIDI input as a "MIDI Controller" in the "Setup"/"Peripherals" configuration:



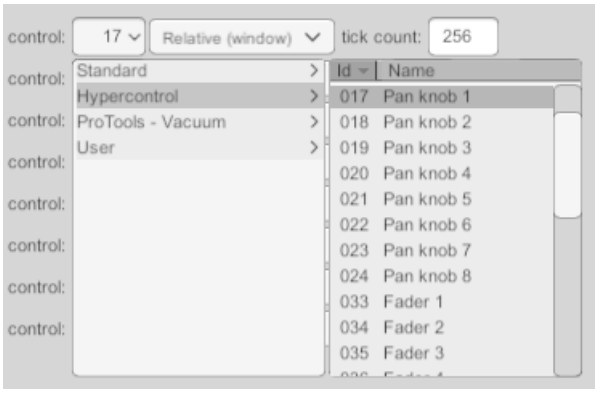
"Type" should be set to "M-AudioKeyboard". "Receive From" is the name of the MIDI port you want to use. "Send To" needs to be set but should be set to a MIDI device that's not being used. Sending it back to the originating port can cause problems so it's best just to send them somewhere that does nothing.

Note that this prevents the MIDI port being used for direct MIDI mapping. Because of this it's best to set up a dedicated virtual MIDI port for this purpose and have another one for regular MIDI CC messages if a mixture of both is desired.

When this mode is active the fader CC numbers directly control the mixer faders. These should be used in Absolute mode. This isn't very optimal and it's recommended that you instead use a fader plug-in for this purpose such as Blue Cat's Gain Suite. Besides being fully mappable these allow you to set the overall volume level manually on the main fader then use the plugin to control the range between 0% and 100%.

To map a input to a plugin control follow these steps:

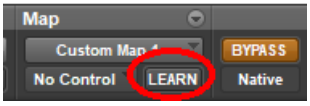
First ensure the Audiate CC output mode is one of the Relative modes ("window" is recommended). There is a section in the "MIDI Control Change Options" page describing how to use these modes. The CC number must be set to one of the "Hypercontrol"/"Pan knob" settings:



Then activate Solo on the desired input in Audiate:



Then in ProTools arm the track as if you were recording to it then click on the "LEARN" option:



Next select the desired control from the drop down list that currently says "No Control". Note that this button is to the **left** of the "Learn" button:

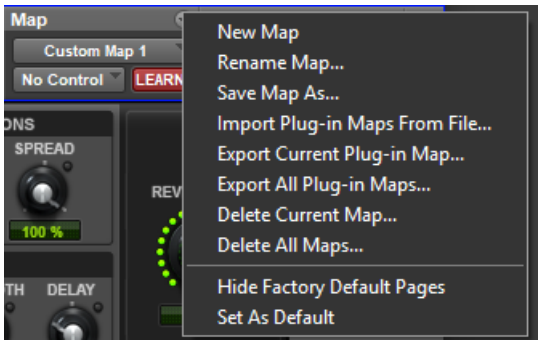
Copyright 2018, 2019, & 2020



The CC number will be learned instantly if it's working. Click "LEARN" to disable the mapping mode. The control should have an indicator showing the same colour that is seen in the Setup/Peripherals dialog:



Once you have mapped the controls in Pro Tools it's worth exporting them to file so they can be restored if changed. This option is hidden away in the little round drop down arrow icon in the top-right of the "Map" panel:



Copyright 2018, 2019, & 2020

Troubleshooting

The following checklist should be checked if a control is not responding or able to be mapped.

System issues:

- Is Steam VR running?
- Are input devices shown as green in the SteamVR Status window?

Audiate issues:

- Is Audiate sending data? Check that the Output Device at the top of the application is showing green.
- Is the virtual MIDI port showing traffic? Many have status displays showing activity.

Application issues:

- Does the application know about the port? Most only scan on startup, ones added after will not be used.
- Is the application instrument listening on that port?
- Is the application instrument listening for CC messages on that port? Ableton for example doesn't by default, you must enable it in the Ableton settings.
- Is the application instrument listening on the same channel?
- Is the track/instrument “armed”? Some applications require this for MIDI events to be received.

Further information on specific applications can be found in the relevant sections of the help pages.

Copyright 2018, 2019, & 2020

Known Issues

4K / UHD Displays

Windows 10 does not scale the application correctly on ultra-high-density displays.

Workaround:

Change the Windows DPI settings used for Audiate. This can be done on a per-application basis as follows:

1. Launch Audiate.

2. Right-click on the entry in the Windows Taskbar, then right-click on the "**Audiate.exe**" entry that pops up.

3. Select "**Properties**".

4. Select the "**Compatibility**" tab

5. Click on "**Change high DPI settings**" near the bottom

6. Check "**Override high DPI scaling behavior**" (also near the bottom)

7. Set "**Scaling performed by**" to: "**System**"

This approach should also work for any other applications you are having issues with.

Copyright 2018, 2019, & 2020

Acknowledgements

This application would not be possible without the work of the following projects:

MidiToolkit

Description: C# MIDI library

URL: https://github.com/tebjan/Sanford.Multimedia.Midi

License: MIT

Zenject

Description: C# Dependency Injection Framework.

URL: https://github.com/svermeulen/Zenject

License: MIT

OpenVR

Description: Virtual Reality API

URL: https://github.com/ValveSoftware/openvr

License: BSD 3-Clause

C#

Description: Programming Language

URL: https://www.microsoft.com/net

License: MIT / Proprietary

Unity

Description: Cross-platform game engine

URL: https://unity3d.com/

License: Proprietary

Vexe Framework (VFW)

Description: Unity development utility, exposes application internals during development.

URL: https://github.com/vexe/VFW

License: MIT

Copyright 2018, 2019, & 2020

Version History

Version 1.0

Initial release.

Version 1.1

Added remote control of patches via MIDI input Program Change messages. Can be used with foot controllers, keyboard inputs and/or pre-configured sequencer events.

Added input device enable button.

Added expand/collapse UI controls.

Improved MIDI device hot-plugging support.

Fixed window resizing issue.

Version 1.2

Added options to automatically expand/collapse unused devices and/or groups of chains when loading patches.

Version 1.3

Added generic device profiles, unknown devices can now use preconfigured profiles for basic support out-the-box.

Added support for custom ID keys used to identify devices, any OpenVR property can be used.

Added support for Windows MR devices.

Added legacy support for Index (Knuckles) controllers. This provides similar inputs as seen with Vive controllers, with the addition of touch sense on the buttons & trigger. Full support will be added in 2.0.

Fixed configuration issue with velocity-based inputs.

Fixed OpenVR property reading issue that could occasionally cause devices to fail to be identified on startup.

Version 1.4

Added additional inputs for touch/click on trigger. Note that touch is emulated on devices without real touch sensors, these require a little pressure to activate.

Improved Oculus support, added additional grip inputs.

Fixed issue on application startup, caused by changes to SteamVR events.

Copyright 2018, 2019, & 2020