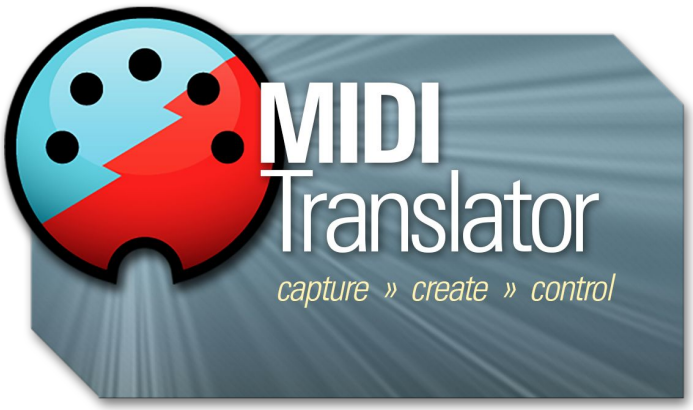
Bome MIDI Translator Pro

USER MANUAL



2025-12-19

Table of Contents

1 Welcome....5
2 Quick Start....9
2.1 Installation on Windows....9
2.2 Installation on macOS....11
2.3 Start MIDI Translator....12
2.4 MIDI Setup....12
2.5 Add a first Translator Entry....14
2.6 Defining Translators....14
3 MIDI Setup Guide....17
3.1 Virtual MIDI Ports....17
3.2 MIDI Devices and MIDI Aliases....18
3.3 Project MIDI Ports....19
3.4 MIDI Router....19
4 Program Interface....20
4.1 Main Window....20
4.2 Toolbar....21
4.3 Menu....21
4.4 The Project....22
4.5 Preset List....22
4.6 Translator List....22
4.7 Properties Sidebar....22
4.8 Event Monitor....23
4.9 Log Window....23
5 MIDI Translator Concepts....25
5.1 Project Level....25
5.2 Preset Level....26
5.3 Translator Level....27
5.4 Incoming Event Processing....28
6 The Project....29
6.1 Author Info....29
6.2 Project Default MIDI Ports....29
6.3 MIDI Ports and Aliases....30
6.4 MIDI Router....30
7 The Preset....31
7.1 Overview....31
7.2 Active vs. Inactive....31
7.3 Always Active....31
7.4 Changing Presets via Outgoing Action....32
7.5 Preset Default MIDI Ports....32
8 The Translator Entry....33
8.1 Overview....33
8.2 Translator Options....33

8.3 Incoming Actions (Trigger)....34
8.4 Rules....35
8.5 Outgoing Actions....35
8.6 Editing Actions....37
8.7 Editing Rules....37
9 Actions....39
9.1 MIDI Message Action....39
9.2 MIDI Port Action....45
9.3 MIDI Router Action (Outgoing)....47
9.4 Keystroke Action....48
9.5 Timer Action....56
9.6 Perform Action....62
9.7 Preset Action....65
9.8 Project Action....67
9.9 Mouse Action (Outgoing)....70
9.10 Execute File Action (Outgoing)....74
9.11 Serial Port Action....77
9.12 AppleScript Action (Outgoing)....83
9.13 Application Focus Action....87
9.14 Variable Storage Action....89
10 Rules and Variables....97
10.1 Overview....97
10.2 Rule Types....98
10.3 Types of Variables....102
10.4 Using Rules and Variables....104
11 Settings....105
11.1 Startup Options....106
11.2 Appearance....106
11.3 Options....107
11.4 Confirm....108
11.5 Virtual MIDI Ports....108
11.6 Serial Port Settings....109
11.7 Export/Import Settings....109
11.8 Reset....110
12 Behind the Scenes....112
12.1 Incoming Event Processing....112
12.2 Executing the Outgoing Action....113
12.3 Parallel Processing....114
13 Tips & Tricks....115
13.1 Make Backups!....115
13.2 Quick Access to Different Configs....115
13.3 Running from a USB Thumb Drive....116
13.4 Timer with 0ms....117

13.5 Leverage the Perform Action....117
13.6 Multiple Actions in one Translator....118
13.7 Performance Optimization....118
13.8 Tips & Tricks in the Bome Forum....122
14 Usage Example....123
14.1 Traktor / Ableton Live Sync....123
15 MIDI Translator in Hardware: the BomeBox....125
16 Reference....127
16.1 Terminology....127
16.2 Keyboard Shortcuts....128
16.3 Command Line Switches....130
16.4 Menu Reference....134

1 Welcome

Thank you for choosing Bome MIDI Translator!

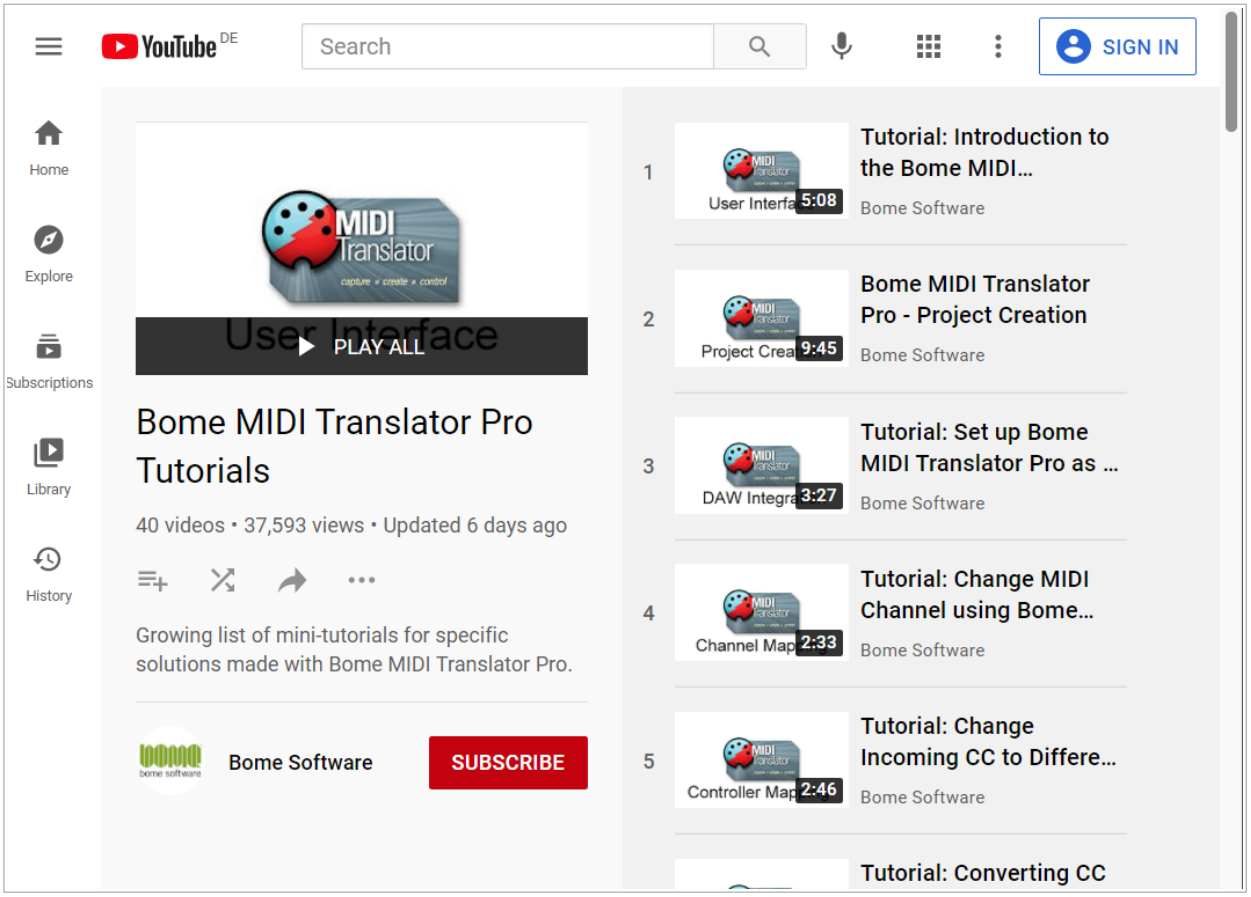
Bome MIDI Translator Pro is a versatile MIDI mapping and translation tool. In addition to working on MIDI messages, you can also translate MIDI to keystrokes, mouse movements, serial ports, and much more.

Best of all, the different translation types can be mixed as you like, so that you can freely redefine your workflow with different devices and the computer as you like.

Here are a few links to get started:

> VIDEO TUTORIALS

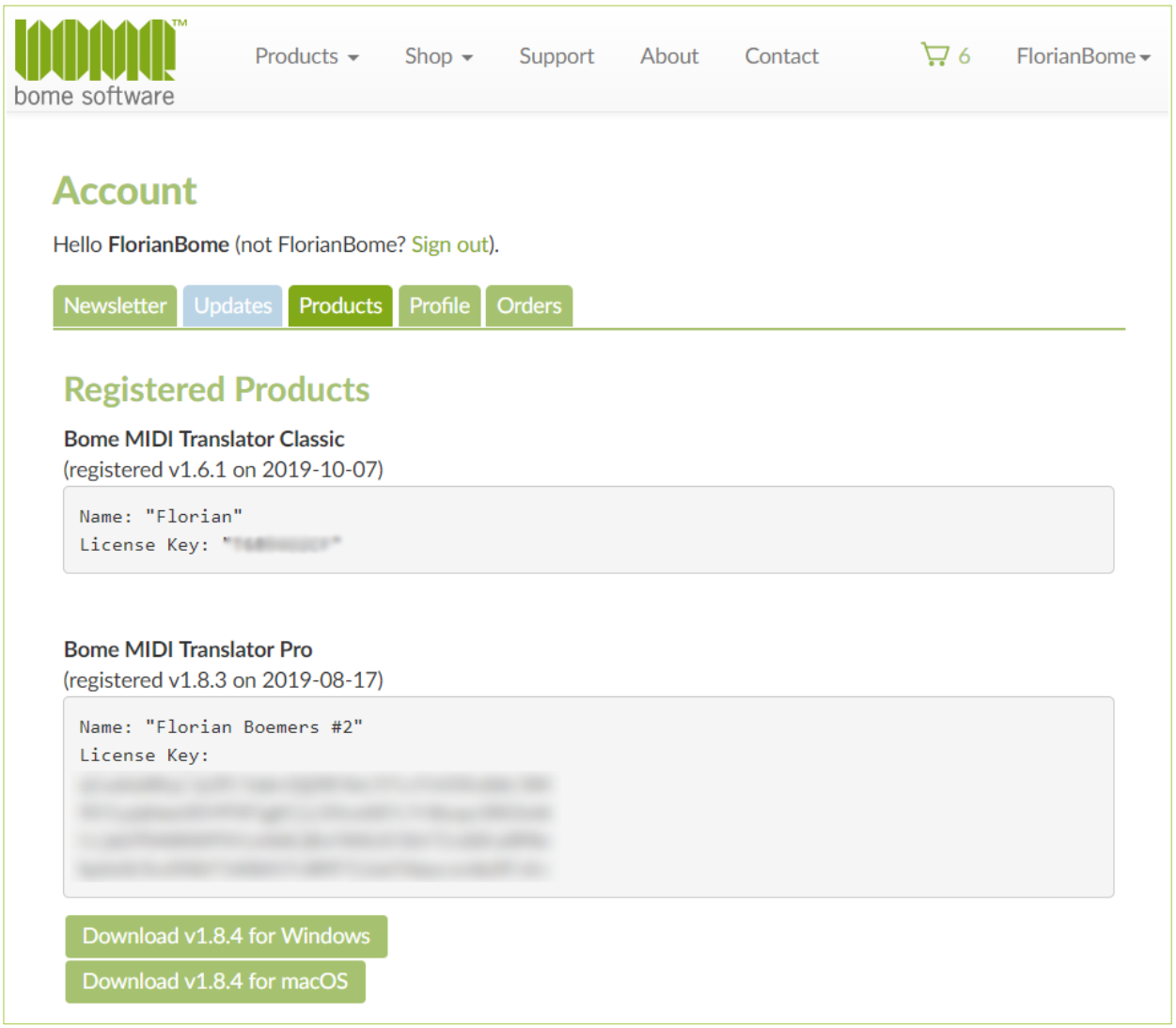
In our YouTube channel, we provide dozens of video tutorials — from simple setup to common solutions, to advanced techniques:



https://www.youtube.com/playlist?list=PLzwHsH6-VZ8Qy_dxsf0LqorjarvnDIFiy

> BOME ACCOUNT

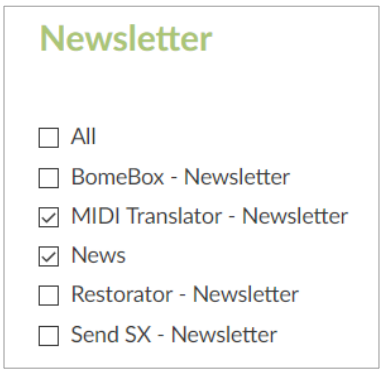
In the Bome Account, you can find all your licenses, downloads, and purchases:



https://www.bome.com/account

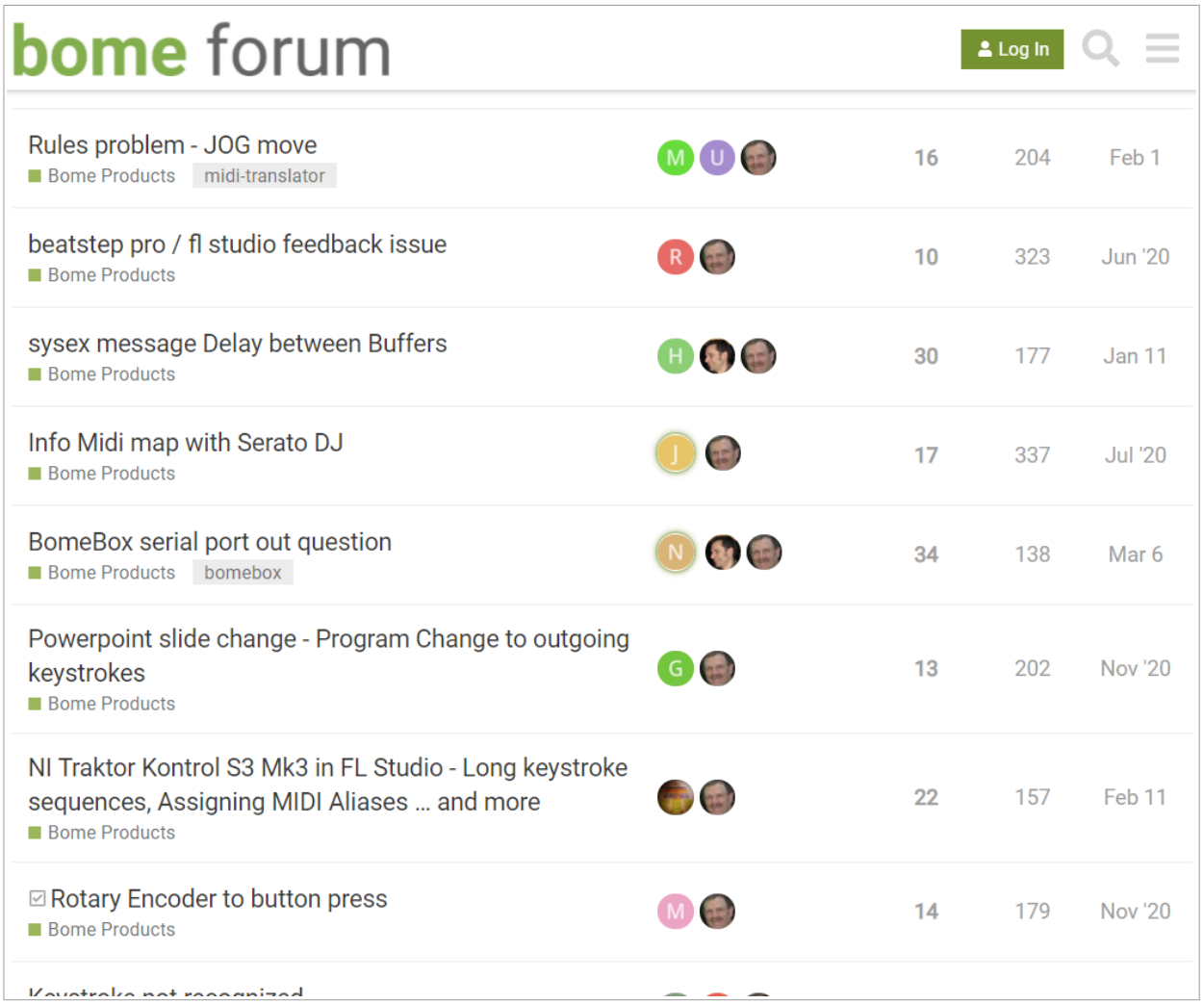
> NEWSLETTER

In order to get update notifications, we recommend signing up to the MIDI Translator newsletter. The volume is very low and after email verification, you can select which news categories you're interested in.



> USER SUPPORT FORUM

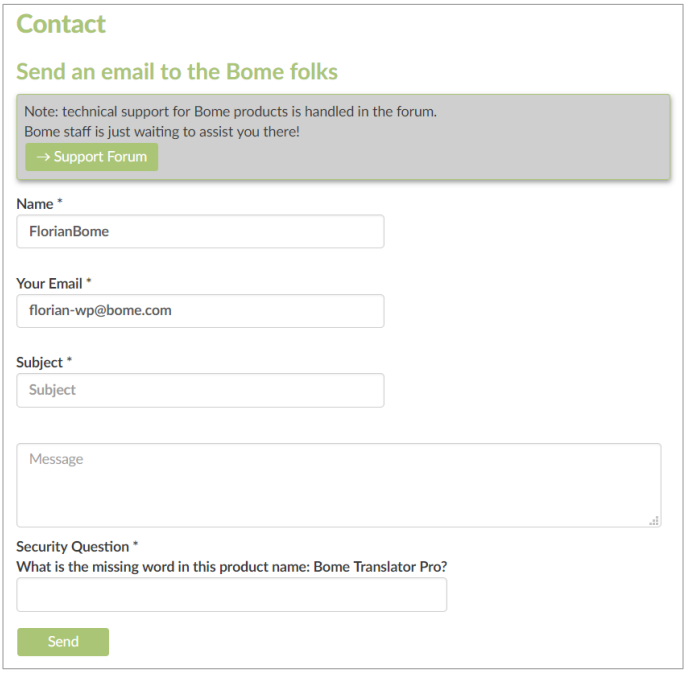
Get professional help for your questions in our Discussion Forum. Our staff is active on the forum and makes sure that every issue receives all the attention it needs.



https://forum.bome.com

> CONTACT US

Feel free to contact us directly in our contact form!



https://www.bome.com/contact

> ENJOY!

Now we hope that you can fully enjoy our software!

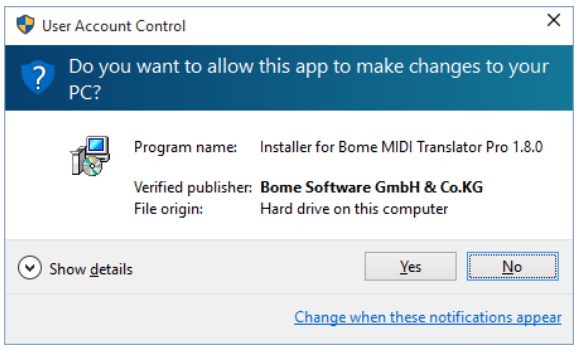
The Bome Software Team

2 Quick Start

2.1 Installation on Windows

> USER ACCOUNT CONTROL

After double-clicking the downloaded installer file, Windows will ask for permission to run the installer.

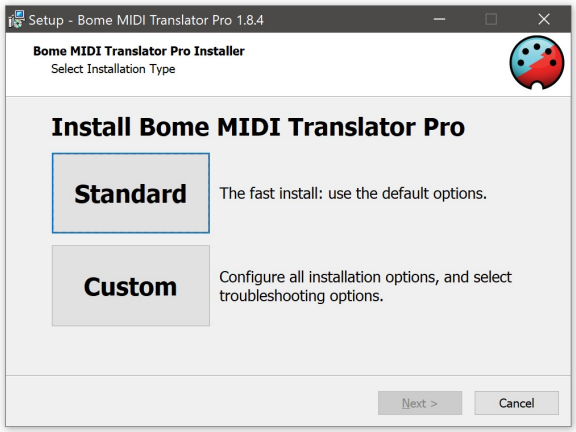


> INSTALLATION TYPE

The initial screen asks you to select the installation type.

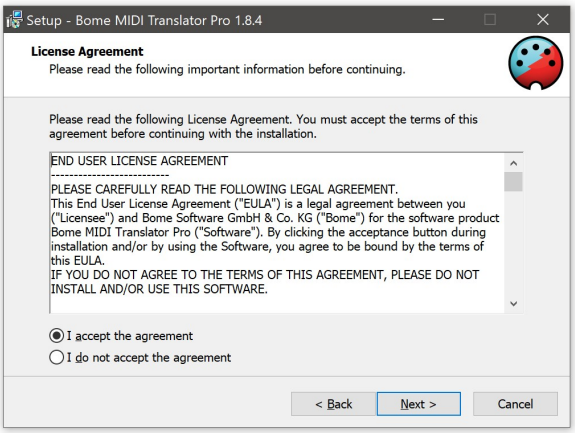
Usually, you'll choose *Standard* here for quick and easy installation with the common options.

For detailed control, use Custom mode. There, you can choose if you want to install the virtual MIDI ports.



> LICENSE AGREEMENT

Please read the license agreement thoroughly. After reading, please select "I accept the agreement", and click *Next* to acknowledge the license agreement and continue with installation.

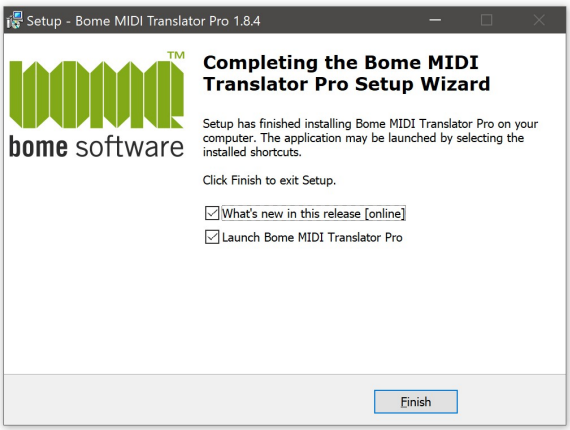


> INSTALLATION

Pressing *Next* will install Bome MIDI Translator Pro on your computer.

> COMPLETION

That's it! You have installed Bome MIDI Translator successfully.



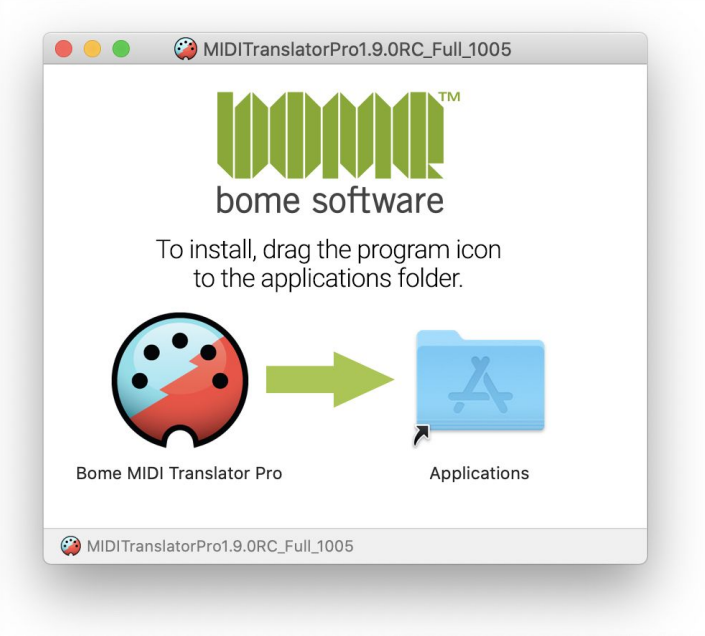
Bome MIDI Translator Pro will start now. You can pin it to the task bar (right-click the task bar icon), to always have it accessible from the task bar.

> STARTING MIDI TRANSLATOR

Start Bome MIDI Translator by double-clicking the desktop icon.

2.2 Installation on macOS

After downloading the .dmg installer file, double click the file: it will open as a volume and display a window similar to the one at right.



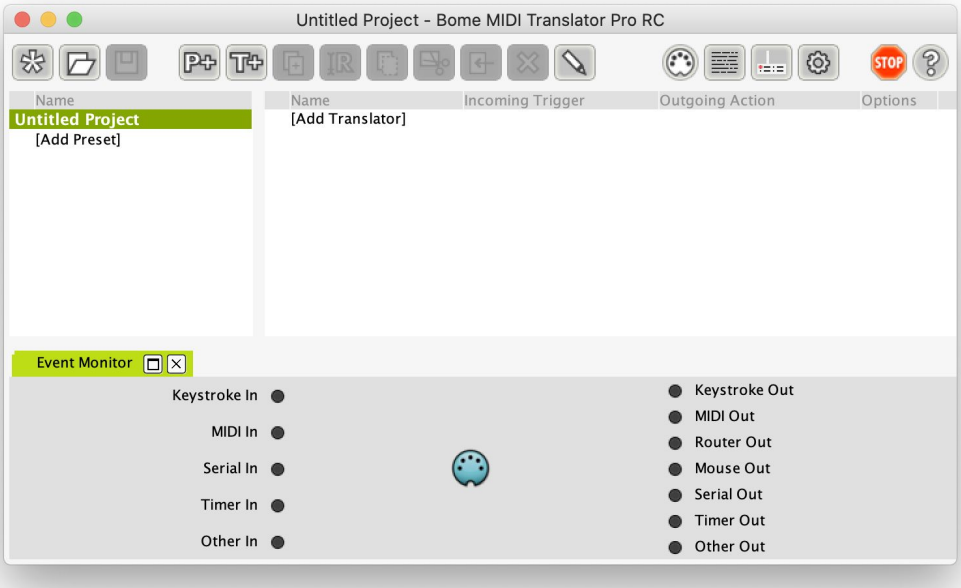
To install Bome MIDI Translator in your Applications folder, simply **drag the Bome MIDI Translator Pro icon to the Applications folder** on the right side.

Then, double-click the Applications folder to start Bome MIDI Translator Pro from there.

Note: if you want to use AppleScript with MIDI Translator, you must install it in the **Applications** folder.

2.3 Start MIDI Translator

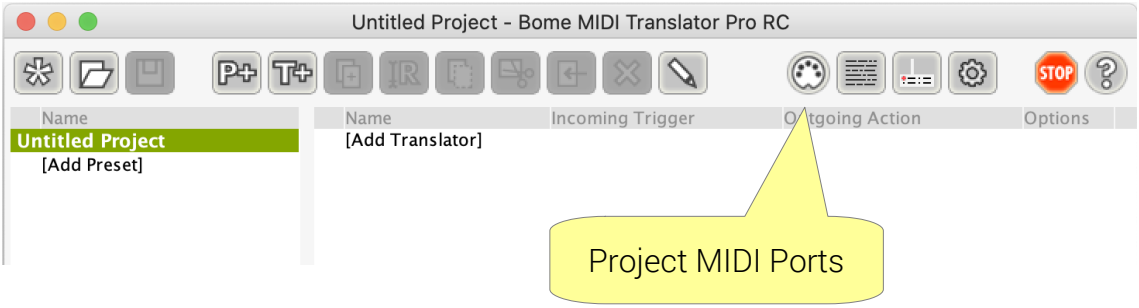
Once installed, start Bome MIDI Translator by double-clicking its icon.



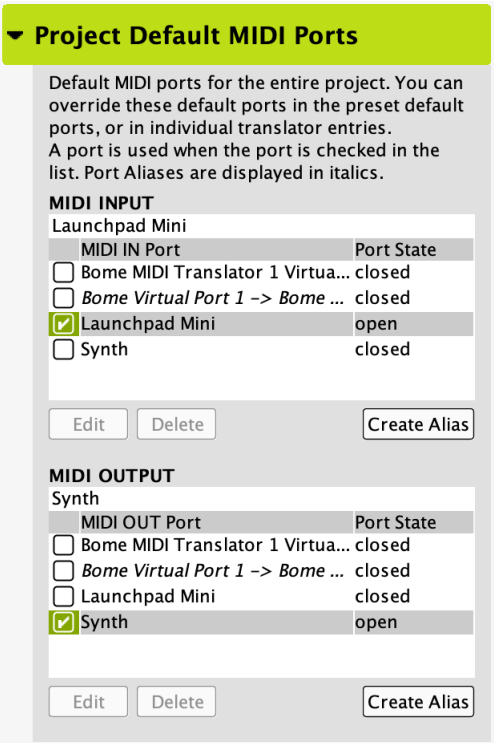
2.4 MIDI Setup

2.4.1 MIDI Settings

The first step in setting up Bome MIDI Translator to work with your MIDI device is to define it in the MIDI settings. To access the MIDI settings, simply press the MIDI settings icon in the toolbar or select the top project filename (*Untitled Project*) on the left.



2.4.2 Define MIDI Ports

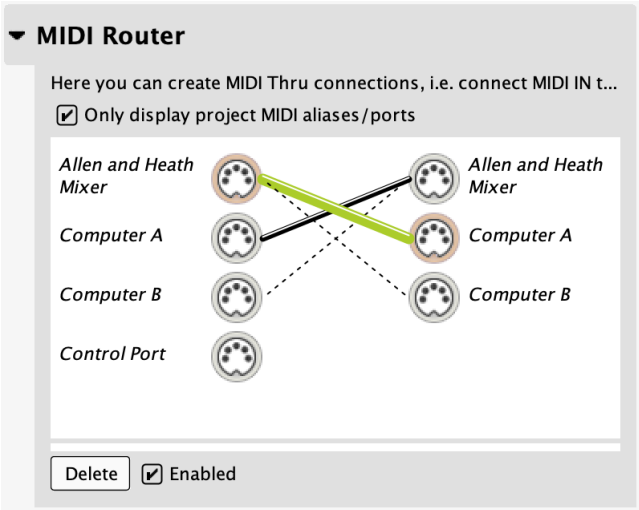


Next, specify the MIDI input and output ports you will be using. The MIDI OUT ports will be the ports to which translated MIDI messages are sent to.

The MIDI IN port will be used as the source of MIDI data, typically connecting with an external MIDI device, e.g. via USB or a MIDI interface on a sound card. Select the appropriate MIDI input source(s) by checking it.

You can use a virtual MIDI port here if you are setting up a translator for a software sequencer or other audio application that interfaces with MIDI. Check the virtual MIDI port as the output device and then select it as the MIDI input port in your 3rd party application in order to have Bome MIDI Translator control it. Use the alias "*Bome Virtual Port 1*" (in italics) instead of the direct device.

2.4.3 MIDI Router / MIDI Thru



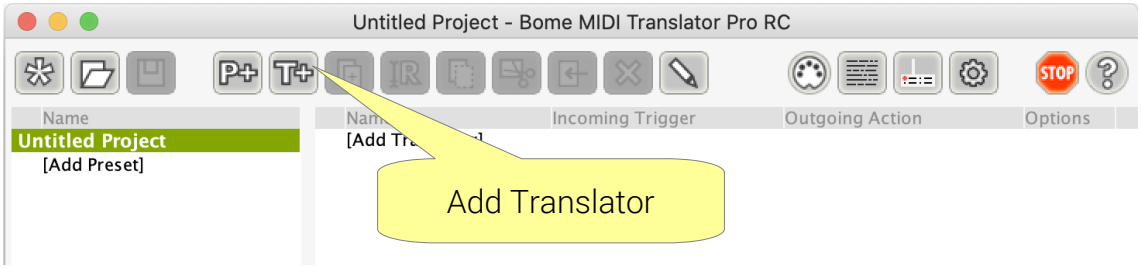
With the MIDI Router, you can create MIDI Thru connections, i.e. connect an input port with an output port. Once connected, all MIDI messages from the input port will be sent directly to the output port. You can use translator entries to add or modify MIDI messages sent to the output port.

Access the MIDI Router in the Project Properties: click on the filename on top of the left list, or use the View menu (keyboard shortcut: Ctrl+3). On the right, scroll down until you see the section titled MIDI Router.

To create a MIDI route, click and drag a MIDI IN connection on the left side of the screen to the desired MIDI OUT connection to enable a MIDI thru connection between the ports. Any data NOT particularly processed by your preset will be routed directly to the designated output port.

2.5 Add a first Translator Entry

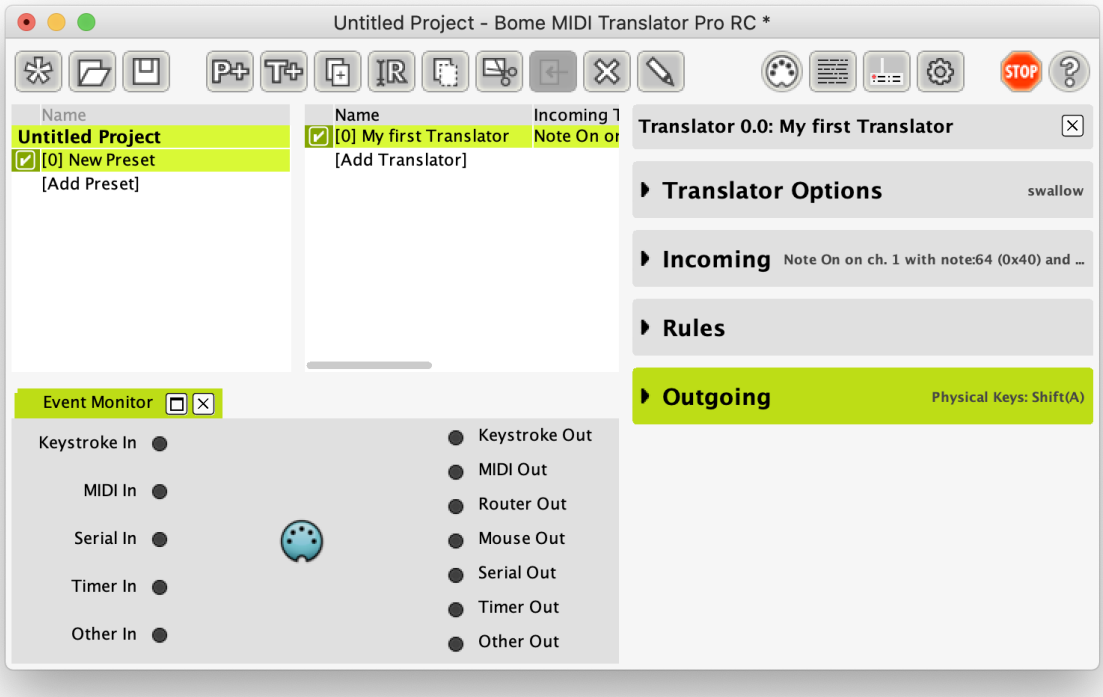
Now, you should have the MIDI interface settings properly configured. Test that they're working correctly by moving a controller on your MIDI device and checking to see if the corresponding light illuminates on the Event Monitor, located at the bottom left.



Now you may begin adding translators. Click the T+ (Add Translator) button on the toolbar to add a new blank translator. Name your translator and press the Enter key. You can now begin working with the translator. If the Properties Sidebar at right is not visible, double click the translator object to enter the Translator Edit screen.

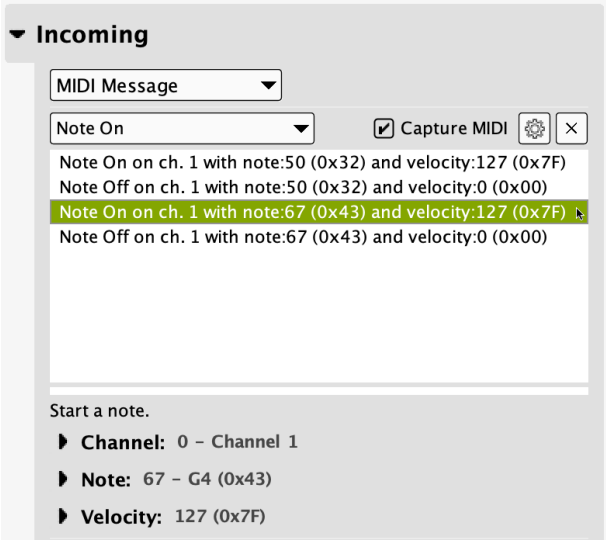
2.6 Defining Translators

Once you've added your first Translator entry, open the Properties Sidebar at right by double-clicking the Translator entry, or by pressing the properties tool button.



The Properties Sidebar at the right is where you specify the incoming and outgoing actions that the translator operates with, as well as the rules and processes that take place between those actions.

2.6.1 Incoming Action



To start, you will need to specify an incoming action to work with. Be sure the Incoming type is set to MIDI message.

Now click the *Capture MIDI* checkbox. Assuming your MIDI settings are correct, you should see a list of MIDI messages scroll by the screen as you move a controller or press a key or button on your MIDI device.

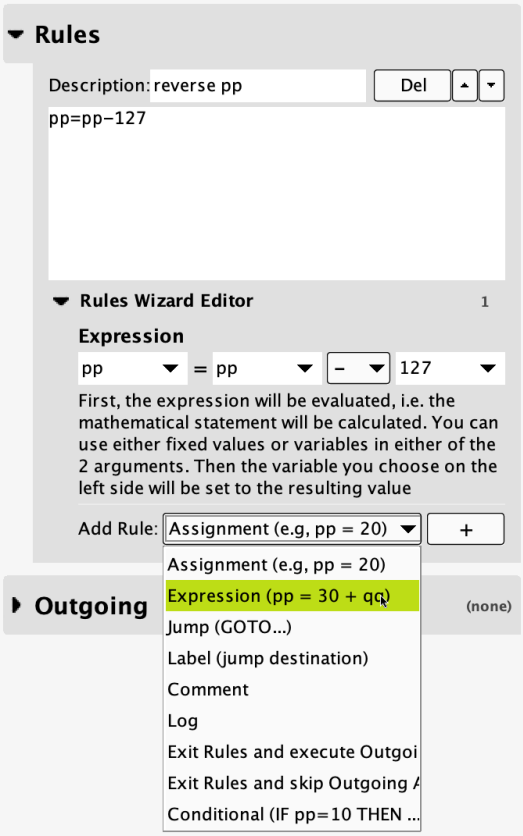
Click on an entry in the Capture list to use it as the incoming trigger.

Press the **X** icon to close the capture list.

You can use the Gear icon next to the *Capture MIDI* checkbox to switch to displaying the received MIDI messages as raw MIDI messages in hexadecimal format. This option is for advanced users. In the gear menu, you can enable/disable timing messages to be included in the captured messages.

See chapter 9 Actions for detailed information on available actions.

2.6.2 Translator Rules



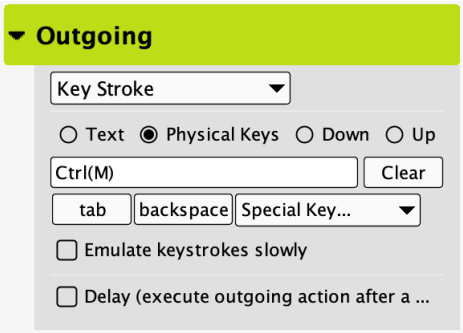
Next, select the Rules tab to view the rules entries for the translator. In this screen, you can specify rules that affect the values local to the translator, or use values stored in global variables.

Click on the Add Rule drop-down list to select the type of rule that you wish to enter. After adding the basic rule, edit the rule parameters with the drop-down lists. Rules can also be entered directly into the text field of the Rules dialog.

In the example, one rule has been entered that will reverse the controller value of a standard MIDI signal. The rule takes the variable value of the incoming MIDI signal (for example, the turning of a

MIDI knob going from 0 to 127) and subtracts it from 127, effectively reversing the signal. Experiment with adding your own rules and editing the rules parameters. See chapter 10 Rules and Variables for more information.

2.6.3 Outgoing Action



Now you need to specify what you want your outgoing action to be for the translator. Select an action type from the drop-down box and enter the action details in the area below. Outgoing action types are varied and depend on the application you are working with and what you are trying to accomplish. Keep in mind that you can use both local and global variables in your translator entries. In the example, we are sending a keystroke in response to the Incoming Action: a keyboard shortcut Command-B.

See chapter 9 Actions for detailed information on available actions.

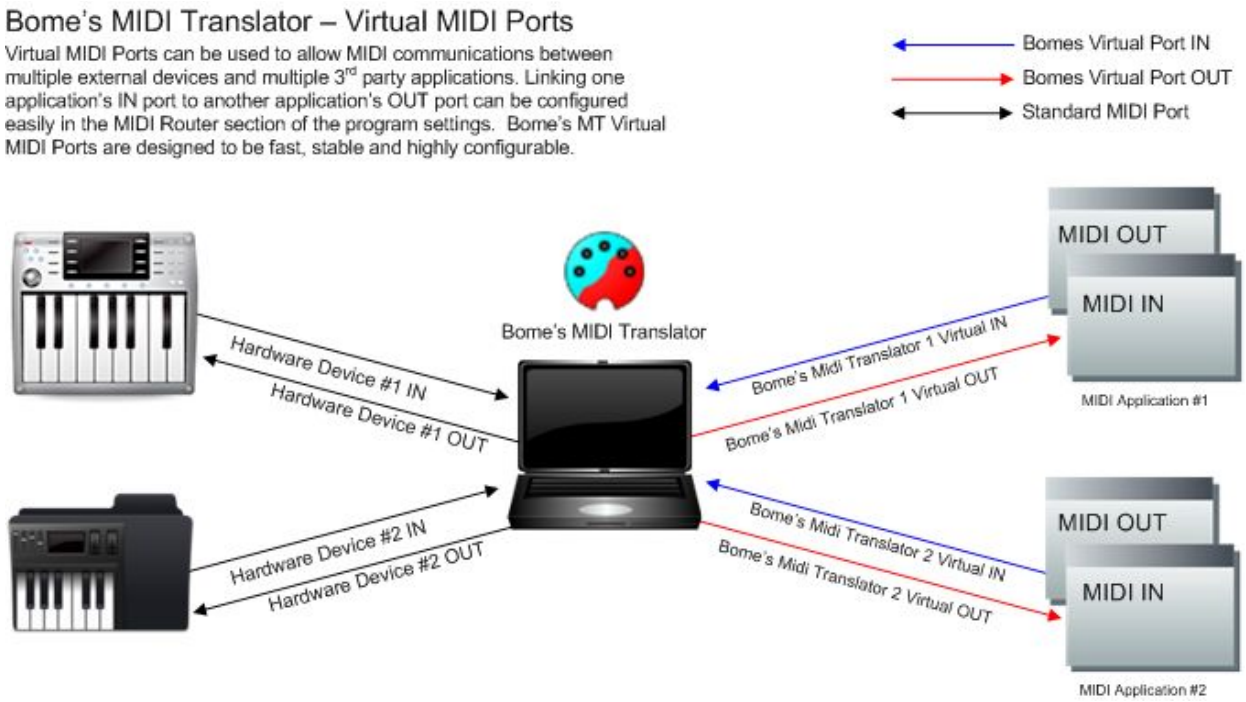
3 MIDI Setup Guide

3.1 Virtual MIDI Ports

A virtual MIDI port driver is included with the application which enables you to send MIDI messages to other programs running on the same computer, and to receive MIDI messages back from them. The virtual MIDI ports in MIDI Translator are unidirectional MIDI ports, requiring the MIDI Translator application running on one end, and any other program destination on the other.

Unlike "loop-back" virtual MIDI ports, which function as both IN and OUT ports simultaneously, Bome virtual MIDI ports only pass through the Windows API once rather than twice to route MIDI data. This added efficiency makes Bome MIDI Translator faster and potentially more reliable, resulting in reduced latency and jitter when operating with MIDI data. Bome MIDI Translator also benefits from a high-speed MIDI data processing engine at its core, delivering MIDI and translator action data at near realtime speeds.

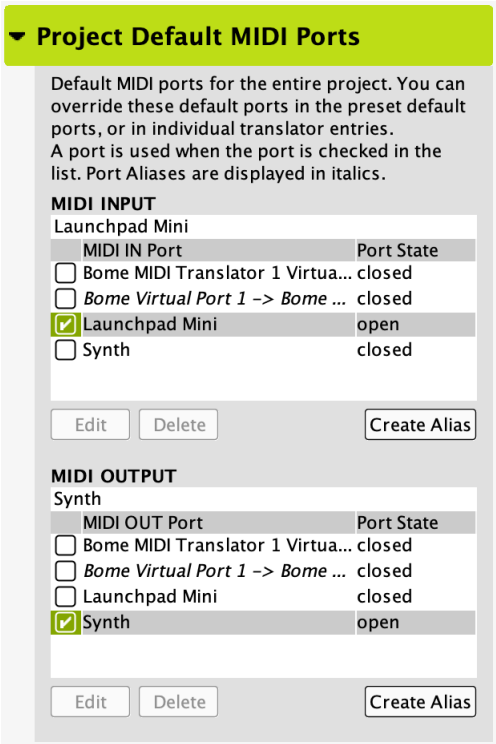
As a result of this direct connection, one end of the virtual MIDI port MUST be connected to the MIDI Translator application via translators or the MT Router. Direct use of the virtual MIDI ports between two other applications is not possible.



virtual port flow diagram

Bome MIDI Translator can communicate directly with any MIDI device or application, allowing it to serve as a powerful hub for MIDI information. A common use for MIDI Translator's virtual ports would be to synchronize the MIDI clocks of two applications. Using MIDI Translator's virtual ports, this is a simple task of linking each application to a Bome virtual MIDI ports, then connecting them in the MIDI Router.

3.2 MIDI Devices and MIDI Aliases



Devices and aliases represent the different MIDI sources and destinations available for MIDI Translator to send and receive MIDI data.

Aliases function as dynamic links to devices, allowing a MIDI Translator project to be shared among users with different MIDI hardware and software. When a Translator entry is created, default input and output MIDI ports are assigned to it based on settings in the Project, Preset and Translator default ports configuration pages. When a project file is opened by a user who has different hardware and software MIDI ports, the program will ask the user to reassign the used ports in the project to different MIDI devices that are available on the computer.

Custom aliases can also be created, allowing you to create named ports in your projects to better organize MIDI communications. For example, it can be beneficial to create named ports in your project such as "KEYBOARD IN" and "TO APPLICATION" to make your projects more human readable.

Default MIDI Ports and/or Aliases can be assigned to different elements of your Bome project, allowing flexibility in routing MIDI data to different devices.

Default ports can be assigned at the overall *Project* level, which will dictate where newly created MIDI translators will receive and transmit MIDI data. After the Project level, default ports/aliases can be assigned at the *Preset* level, allowing you to override the Project default ports and have entire Presets dedicated to managing the MIDI data coming from or going to a particular device or devices. Lastly, individual translator entries can have specific port assignments that override both the Preset and Project default ports.

- Devices represent actual hardware and software MIDI ports
- Aliases are pointers to these devices which can be reconfigured to point to other devices on the fly.

- Use aliases to create human-readable names for your MIDI sources and targets
- Your project file stores the default aliases you're using in your project.
- Aliases are displayed in *italics*.
- For every virtual MIDI port, there is an automatically created alias. You should use the alias for best portability of your project file.

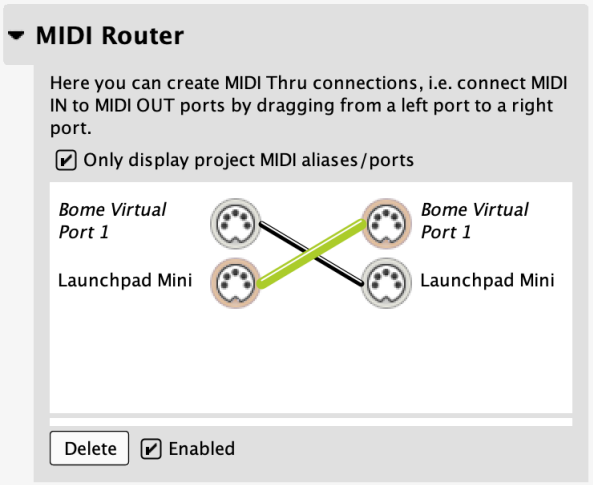
3.3 Project MIDI Ports

MIDI Translator Pro has the concept of *Project MIDI Ports*. These are the ports that you use in the project:

- Selected in the Project Default MIDI Ports
- Selected in any preset's Default Preset MIDI Ports
- Selected in an Incoming MIDI Action or Outgoing MIDI Action
- Used in a MIDI Route

3.4 MIDI Router

By default, MIDI Translator does not route any MIDI data. For MIDI data to be processed, either a Translator must be created for it, or a MIDI Router connection must be made.

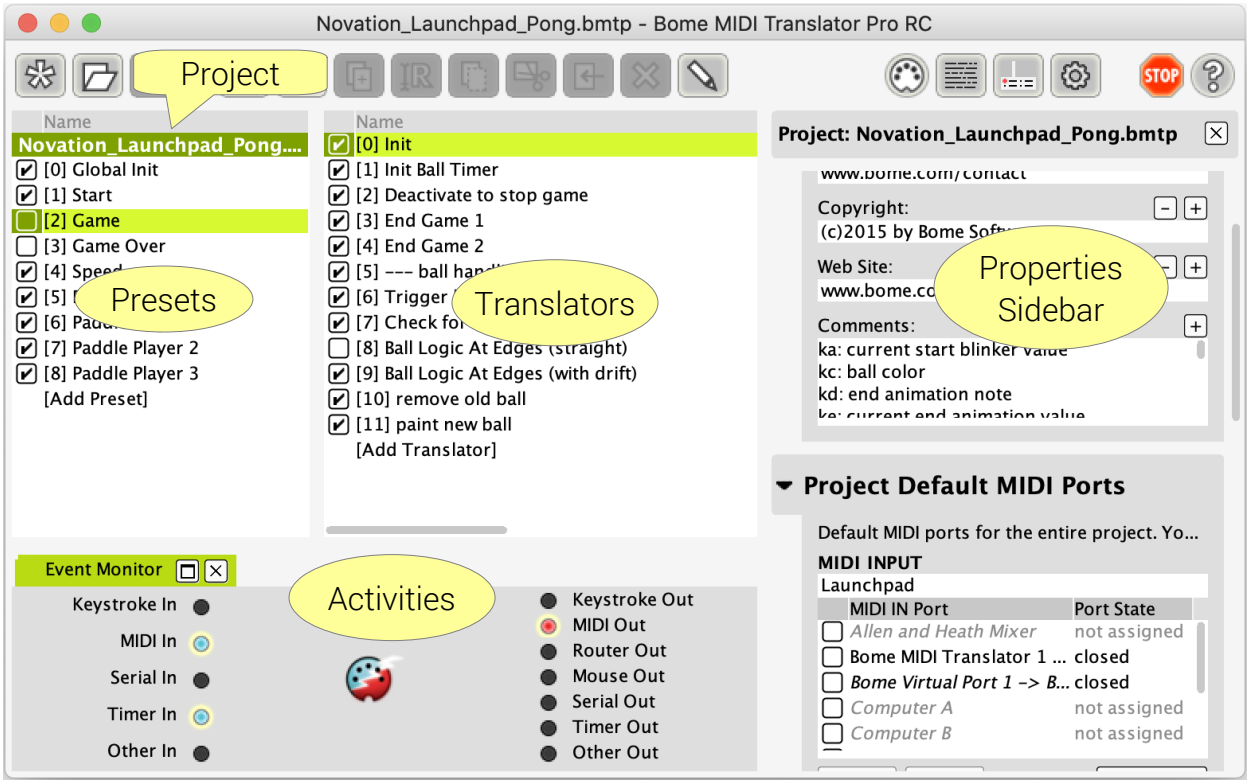


The MIDI Router is a patch panel type setup screen that allows for "patch cords" to be created between available MIDI devices and aliases. Multiple connections can be made from a single source MIDI IN device allowing MIDI data to be replicated and sent to multiple devices concurrently.

A MIDI Router patch connection in Bome MIDI Translator can effectively be thought of as a MIDI Thru connection. Any MIDI data that is received by a source device in a MIDI Router patch connection is retransmitted to all connected destination devices.

4 Program Interface

4.1 Main Window



From the main window of Bome MIDI Translator, a user can manage projects, presets and translators from start to finish. The main interface is subdivided into five main sections: the menu/toolbar, the preset list, the translator list, the Properties Sidebar, and the activity area.

- Project: clicking the project filename will activate the Project Properties.
- Presets: Collections of Translator objects that can be activated or deactivated easily
- Translators: capture Incoming Triggers (which can range from MIDI messages, keystrokes, event timers). Captured data can then be manipulated and retransmitted as a different MIDI message, or an entirely different action.
- Properties Sidebar: depending if Project, Preset, or Translator is selected, display the corresponding properties/editor.
- Event Monitor: view realtime activity of sources, targets, and internal processing
- Log Window: text output of what MIDI Translator is doing (not visible in screen shot above)

4.2 Toolbar

The Toolbar enables easy access to the most commonly used menu items in Bome MIDI Translator. There is an equivalent menu item for each of these icons, which can be accessed via the menu or via keyboard shortcut.

Icon	Name	Description
	New Project	Start a new empty project
	Open Project	Open a Project file from disk (Ctrl+O / Cmd+O)
	Save Project	Save the current Project to disk (Ctrl+S / Cmd+S)
	New Preset	Create a new Preset (Ctrl+Shift+P / Cmd+Shift+P)
	New Translator	Add a new Translator (Ctrl+Shift+T / Cmd+Shift+T)
	Duplicate	Create a new Preset or Translator as a copy of the currently selected Preset or Translator (Ctrl+D / Cmd+D)
	Rename Preset	Rename selected Translator or Preset (F2)
	Copy	Copy the selected Translator(s) to the clipboard
	Cut	Cut the selected Translator(s) to the clipboard
	Paste	Paste Translator(s) from the clipboard
	Delete	Delete selected Translator(s) / Preset(s) (DEL)
	Properties	Show and select the Properties Sidebar with Translator, Preset, or Project properties (ENTER)
	MIDI	Go to MIDI ports in the project properties
	Log Window	Show the Log Window in the lower left area
	Event Monitor	Show the Event Monitor in the lower left area
	Settings	Open the Settings window
	Stop	Reset the MIDI out device (Panic) (Shift+ESC)
	Help	Show Help Topics (F1)

4.3 Menu

There are 5 main menu items:

- File—operations on the file and the entire project, like open/save, restart, etc.
- Edit—operations on the currently selected Preset or Translator
- MIDI—MIDI settings and options
- View—show/hide sub-windows
- Help—show this user manual, link to program update and support

4.4 The Project

Pressing the filename at top of the Preset List will activate the project. If the Properties Sidebar at right is visible, it shows the project properties.

4.5 Preset List

Available presets are listed on the left, in the preset list.

A preset is a collection of translator entries. You can create as many presets as you like, there is no functional difference if you create 10 presets with 1 translator each, or 1 preset with 10 translators.

Each item in the preset list has an appropriate context menu that is easily accessed by right-clicking a preset entry.

Use the check boxes to activate/deactivate the presets. Once a preset is selected, the right Properties Sidebar will show the properties of the selected preset. The edit functions like Duplicate, Copy, Paste, etc. will work on the selected preset.

4.6 Translator List

To the right of the preset list is the translator list which contains the various translator entries that are defined in the selected preset, along with a check box for activating/deactivating the entries and a brief rundown of the incoming and outgoing actions for each.

Each item in the translator list has an appropriate context menu that is easily accessed by right-clicking on an entry. Also, global-level actions for translators can be accessed by right-clicking the list background.

If the Properties Sidebar is visible, selecting a Translator will show its properties in the Properties Sidebar. You can double-click a translator or press ENTER to activate the translator properties.

4.7 Properties Sidebar

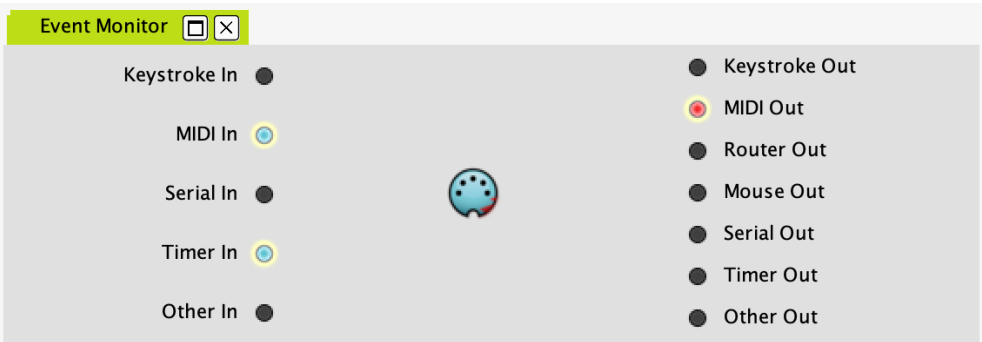
When activated, Properties Sidebar takes the entire right side. It displays editors for the currently selected project, preset, or translator.

To invoke Properties Sidebar, press ENTER in either the project, preset list, or translator list, or double click an item in the lists, or press the properties toolbar button: . You can also quickly view and activate the project properties by pressing Ctrl+3, or select a preset and jump to its properties by pressing Ctrl+4. For editing the translator properties, use these shortcuts: General Ctrl+5, Incoming Ctrl+6, Rules Ctrl+7, Outgoing Ctrl+8.

Last, but not least, the project default MIDI ports are shown and selected when you invoke the MIDI toolbar button .

4.8 Event Monitor

When active, the lower left activity area shows the event monitor, where you can quickly see what internal and external signals Bome MIDI Translator is processing. The virtual LEDs will flash when the noted action is being performed, or the noted signal is being received or transmitted.

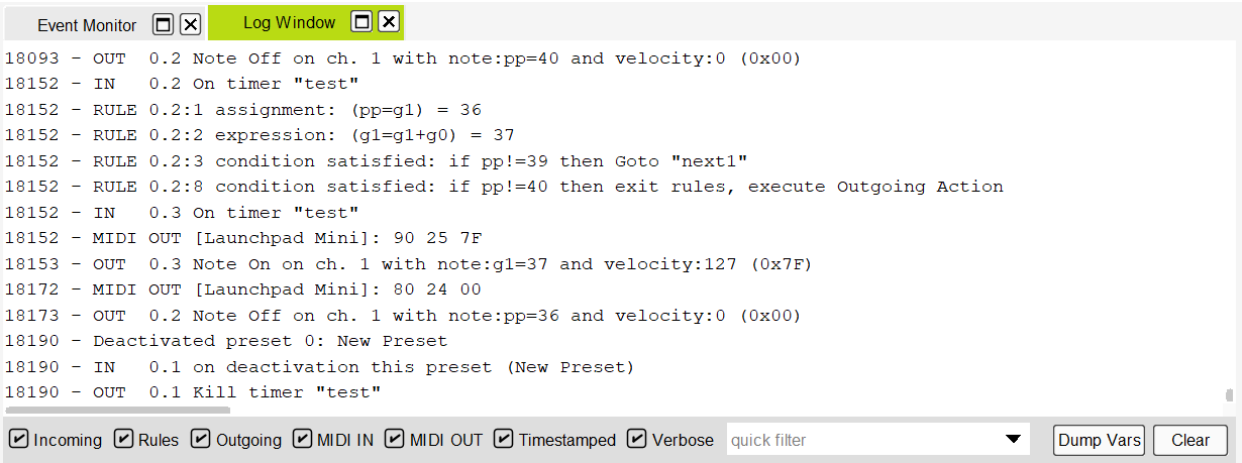


Event Monitor

You can activate the event monitor in the View menu, or with the toolbar button.

4.9 Log Window

The Log Window shows a continuous text stream with real time notifications what the MIDI Translator engine is currently doing.



The Log Window is a great help when developing complex presets with rules and other logic.

Use the **checkboxes** at the bottom to select the types of events to display and whether you want the displayed events to include a time stamp.

Incoming	If checked, log any Incoming Trigger which was triggered by an incoming event.
Rules	If checked, log execution of every Rule in executed Translator entries. Useful when developing Rules, but can quickly fill up the log window with a lot of Rule log messages.
Outgoing	If checked, log any execution of an Outgoing Action.
MIDI IN	If checked, monitor every message that is being received by an open MIDI IN port.
MIDI OUT	If checked, monitor every message that is being sent to an open MIDI OUT port.
Timestamped	If checked, every log message is prepended with a millisecond timestamp.
Verbose	If checked, the engine will log more detailed information.

Enter a search term into the **quick filter box** to only show lines which contain the search term.

Note that displaying many entries in the Log Window can use a lot of CPU resources and affect real time performance of the MIDI Translator processing engine. For live use, disable the Log Window.

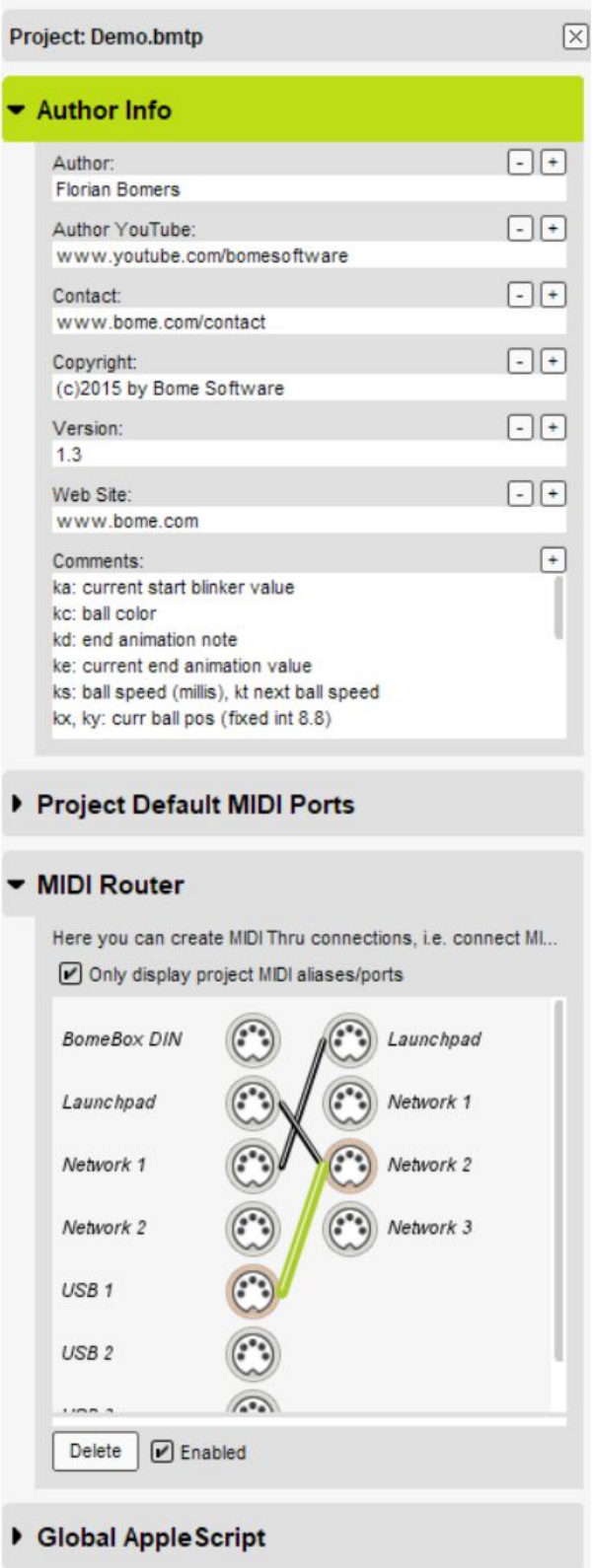
Clicking the **Dump Vars** button will list all global variables in the Log Window which have a value other than 0.

Click	List global variables: values are displayed in decimal form.
Ctrl+Click Command+Click	List global variables: values are displayed in hexadecimal form.

To see the Log Window, use the toolbar button or invoke it from the View menu.

5 MIDI Translator Concepts

This chapter outlines the main concepts of MIDI Translator's concept. Refer to the following chapters for in-depth descriptions of the respective topics.



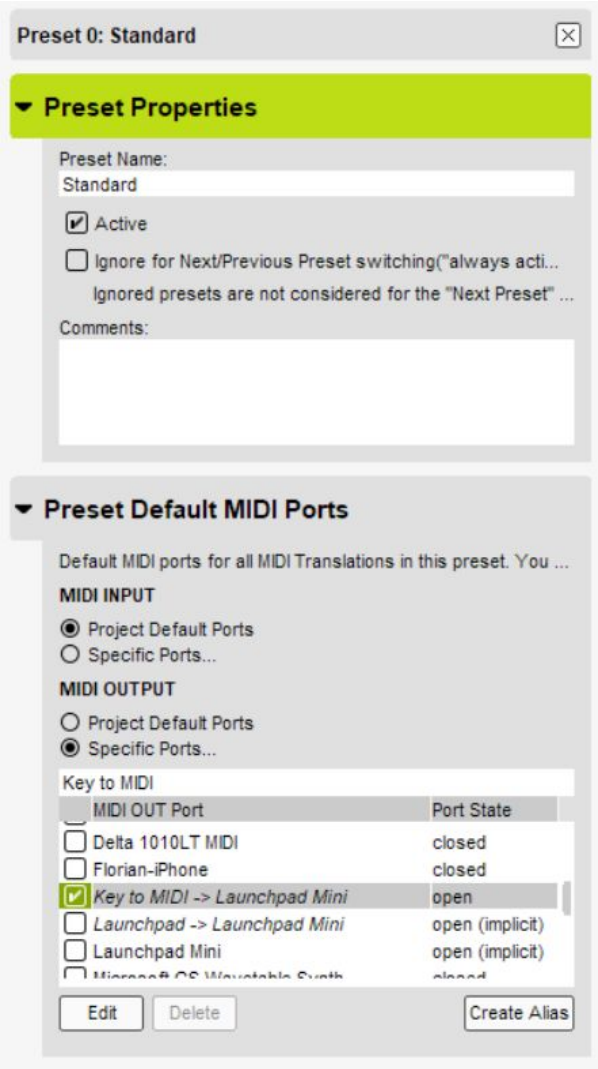
5.1 Project Level

In MIDI Translator, a Project is equivalent to a .bmtp file that you can load and save from within MIDI Translator. A Project includes the following sections:

- **Author/Project Info**
  name, web site, and other info of the author of this project
- **Project Default MIDI Input ports**
  Set the MIDI devices (aliases) that this project receives from. If you don't define specific MIDI Input ports at the Preset or Translator level, the incoming MIDI actions will receive from these MIDI ports.
- **Project Default MIDI Output ports**
  Set the MIDI devices (aliases) that this project sends to. If you don't define specific MIDI Output ports at the Preset or Translator level, the outgoing MIDI actions will send to all the ports selected here.
- **MIDI Router**
  Define MIDI Thru connections for the entire project.
- **Global AppleScript**
  Here you can define AppleScript handlers which you can call from outgoing AppleScript actions. See the AppleScript chapter for more details.
- **Presets**
  The Project owns all presets which you can see in the Preset List.

5.2 Preset Level

A Project can have one or more Presets, they are always visible in the Preset List at left.



A Preset is a collection of Translators. You can deactivate a Preset by unchecking the *Active* check box in the Preset List or with the equivalent checkbox in the Preset general properties. All Translator entries in an inactive Preset are ignored during event processing. It is possible to activate/deactivate Presets by way of outgoing actions in a Translator.

A Preset contains the following:

- **Preset Properties**
  Name, active/inactive, *Always Active* convenience setting
- **Preset Default MIDI Input ports**
  Set the MIDI devices (aliases) that the Translators in this Preset receive from by default. If you don't define specific MIDI Input ports in a Translator, it will receive MIDI only from the ports selected here.
- **Preset Default MIDI Output ports**
  Set the MIDI devices (aliases) that the Translators in this Preset send MIDI to by default. If you don't define specific MIDI Output ports in the Translator's outgoing action, it will send to all MIDI ports selected here.
- **Translators**
  The preset owns a list of translator entries, as visible in the center Translator List.

5.3 Translator Level

A Translator is the work horse of your project: here you define the translation conditions and reactions. You can add as many Translators into a Preset as you like. The Translator List only shows the Translators in the selected Preset.

A Translator has the following 4 items:

5.3.1 Translator Options

General settings: name, active/inactive, stop processing.

5.3.2 Incoming Trigger

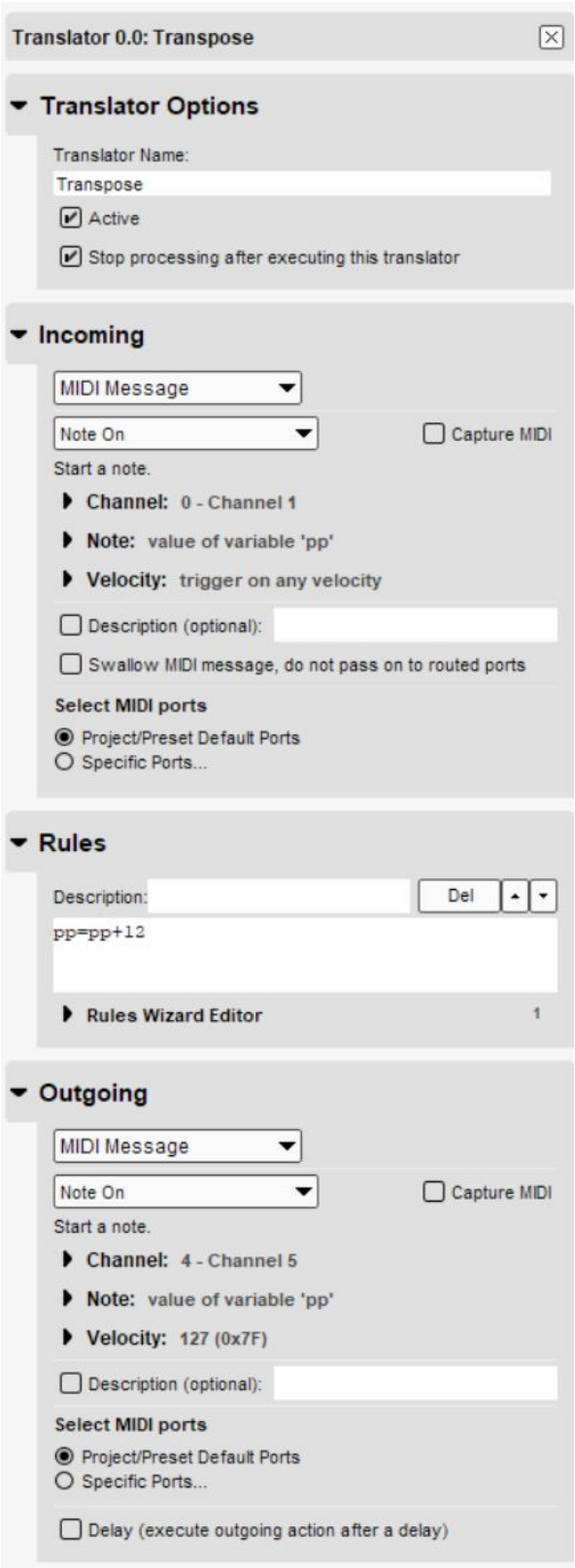
The condition for processing this Translator. You can choose from a variety of incoming trigger types, like MIDI messages or typed keystrokes. There are also MIDI Translator internal events like when a Preset is activated or when a Timer interval elapses.

5.3.3 Rules (advanced)

If the Incoming Action is triggered by an event, the Rules are executed. Rules are simple logic and math statements for advanced usage.

5.3.4 Outgoing Action

If the Incoming Trigger was triggered (and the Rules don't cancel the Translator action), the Outgoing Action is executed. There are many different action types: for example, you can send a MIDI message, or emulate typing a keystroke. You can also affect MIDI Translator's internal behavior by activating or deactivating Presets, or by starting a Timer.



5.4 Incoming Event Processing

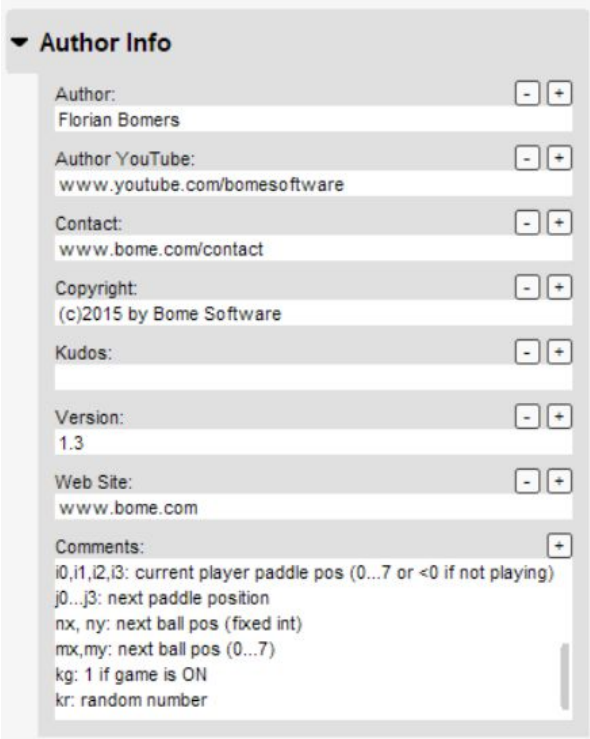
When MIDI Translator receives an incoming event, it starts to process it with the first Translator entry in the first Preset (provided that the Preset is activated). Then it continues on to the second Translator in the first Preset, and so on until all Translators in the first preset have processed this incoming event. Then the same is repeated for the second Preset. This is done until the event is processed by all Translators in all Presets.

If during event processing a translator entry has the *Stop Processing* flag set and is triggered by the event, processing of this event is interrupted and any following translators and presets will not see this event.

For a more in-depth explanation of the engine's event processing, see chapter 12.1 Incoming Event Processing.

6 The Project

6.1 Author Info

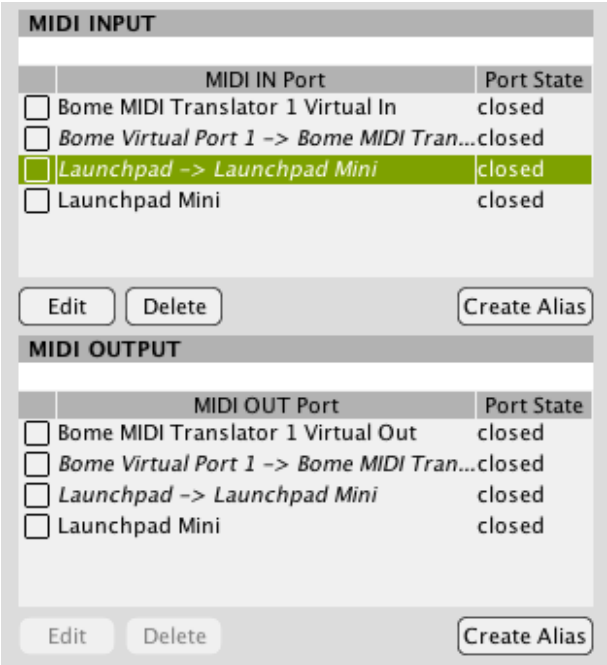


The Author Info screen lets you attach information to your Bome MIDI Translator project file that will travel with the file if you decide to redistribute it. This section is particularly of use if you are sharing project template files with other users.

You can also use the Author Info fields to keep comments to yourself about the organization of this project, like variable usage.

Use a + button to add an author info field, or the – button to delete a respective info field.

6.2 Project Default MIDI Ports



In the Default MIDI Ports, you can specify the incoming and outgoing MIDI ports that are used by the project when it is opened. This functionality is useful when transporting project files between computers that may have different MIDI controllers. Project MIDI port aliases can be created in the MIDI Ports / Aliases screen and selected in the Default MIDI Ports screen to ensure that rules created for one MIDI device can be linked to another easily.

The Default MIDI Ports screen can be accessed via the program menu by navigating to File / Project Properties / Default MIDI Ports, from the View menu, or by selecting the MIDI toolbar button:

6.3 MIDI Ports and Aliases

The MIDI Ports list is where you set your incoming and outgoing MIDI ports, as well as where you specify project aliases. Simply mark the checkbox next to the MIDI IN and OUT ports you wish to use in your project and they will become available for use in translators.

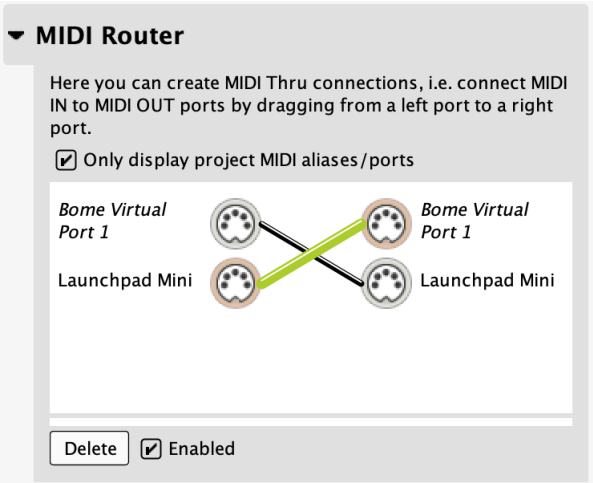
Any project that is opened from another user with different MIDI hardware will STILL have entries for their unique MIDI incoming and outgoing settings. Using the MIDI ports list(s), one can reassign the MIDI assignments of a project to point to any combination of hardware and/or software MIDI ports.

To create a new MIDI port alias, simply click on the *Create Alias* button under the MIDI port listing. A new alias will be created, which can be named anything and assigned to any hardware or software MIDI port.

MIDI Aliases are useful for working with multiple input and output MIDI sources, such as connecting multiple hardware MIDI devices, or connecting one or more hardware MIDI devices to multiple software inputs.

6.4 MIDI Router

The MIDI Translator Router is a powerful but simple way for MIDI Thru connections to be made between MIDI Interfaces. All detected MIDI IN ports and aliases are displayed on the left column of the screen, while MIDI OUT interfaces and aliases are displayed on the right side. Simply drag and drop a line between the two ports you wish to create a THRU connection between and one will be created, represented by a solid line connecting them.



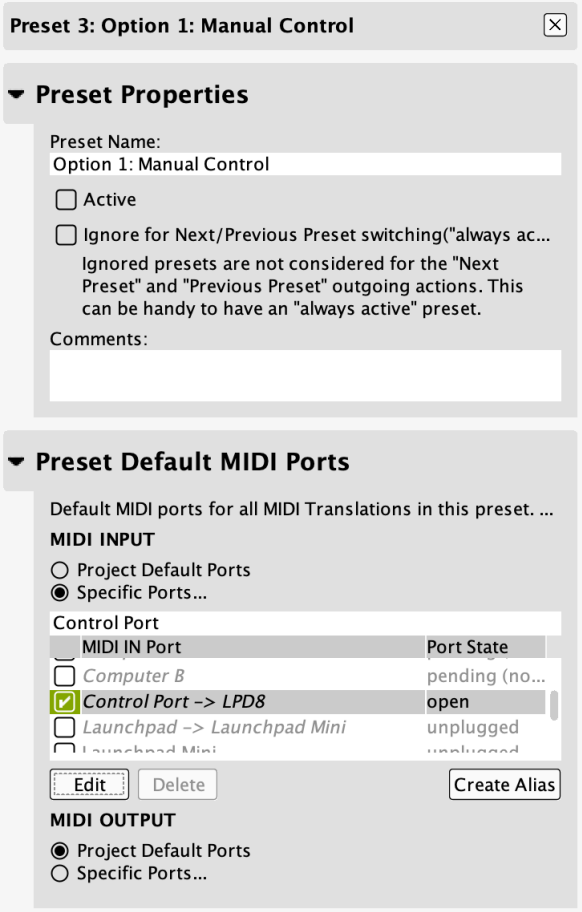
MIDI THRU connections can span from one MIDI IN port to many MIDI OUT ports. This will effectively duplicate all MIDI messages to the connected MIDI OUT ports.

Conversely, you can create connections from multiple MIDI IN ports to a single MIDI OUT port. This results in MIDI messages being merged.

To access the MIDI Router, select it from the MIDI menu, or scroll down in the project properties.

7 The Preset

7.1 Overview



Bome MIDI Translator encapsulates Translator entries into 'Presets' which can be managed at a more detailed level than a setup where every translator is active all the time.

The far left list in MIDI Translator Pro is the Presets list. Select a Presets by clicking its name in that list. Now the Translator List in the center will show all Translator Entries belonging to that Preset. The Properties Sidebar at far right shows the properties of the selected Preset.

7.2 Active vs. Inactive

A preset's active or inactive state can be determined by looking at the checkbox next to its name, or the Active checkbox in the Preset Properties. If the checkbox is checked, the preset is active and its Translators are being processed. If the preset's checkbox is unchecked, it is deactivated and no processing occurs with the Translators in this Preset.

When the Preset List is focused, you can also use the space bar for toggling the active state. Presets can also be managed via the context menu accessed by right-clicking either the preset itself (copy, rename, delete, etc) or by using the Edit toolbar buttons.

Last, but not least, Presets can also be activated and deactivated via Outgoing Actions in Translator entries.

7.3 Always Active

Checking the *Ignore for Next/Previous Preset Switching (Always Active)* property for a Preset will render it exempt from the rules of Next/Previous Preset Change Outgoing actions. When it's *always active* it will not be touched when using the “Next Preset” or “Previous Preset” Outgoing Actions.

7.4 Changing Presets via Outgoing Action

Presets can be activated dynamically via an Outgoing Action from within your Translators. See section Preset Action on page 65 for details.

7.5 Preset Default MIDI Ports

Every Presets can have its default MIDI ports defined. Default Input MIDI ports are useful if you have multiple MIDI IN devices you wish to manage independently from one another, or you have MIDI hardware on the same MIDI channel that you wish to separate. Default MIDI Output ports are useful if you are working with multiple software programs or outboard devices, and you wish to divide and manage translator data among them.

Defining Default Ports for a preset is simple. First, select the Preset you wish to change, then check the properties for the Default MIDI Ports sections. All the ports that you check will be defined as preset default ports.

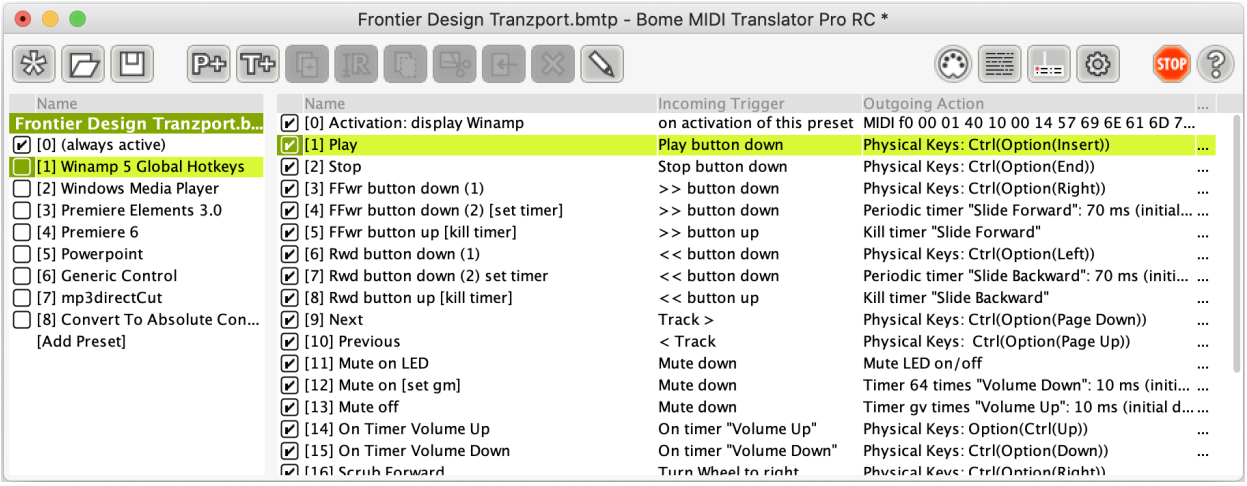
Also note that you can select "Project Default Ports" to use the default project ports as defined in the Settings screen. Preset Default Ports override Project Default Ports.

We recommend selecting MIDI devices exclusively on Preset level.

8 The Translator Entry

8.1 Overview

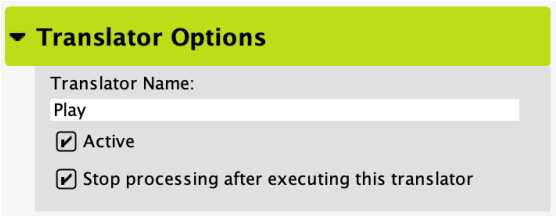
Translator Entries comprise the core functionality of Bome MIDI Translator. In simple terms, Translator Entries listen for an Incoming Action, optionally do some processing on the Incoming Action (see the Rules section of the manual), and then optionally output an Outgoing Action. Translators are limited to a single Incoming Action and Outgoing Action, but you can easily create multiple Translator entries with identical Incoming or Outgoing Actions depending on your needs. Translators can, however, transmit and receive on multiple MIDI ports, making interfacing with different devices easier. Also, the routing flexibility of outgoing actions and rules allow for a lot to be accomplished with a single translator.



In the screenshot above, the *Play* entry is the currently selected Translator entry. This Translator listens for the Play button on the MIDI device. Once pressed, it emulates a keystroke combination: `Ctrl+Option+Insert`.

8.2 Translator Options

There are three main settings in the 'Options' section in the translator properties.



8.2.1 Name

This is the simple descriptive property of the translator. It does not have any function other than for reference in Presets. The Translator name does not need to be unique, therefore multiple translators can have the same name. It is recommended that the Translator name be something simple that will make identifying multiple translators in large presets more easy. You can also edit the name directly in the Translator List (shortcut F2).

8.2.2 Active

This option determines whether the translator is actively being processed (listening for defined Incoming Action) or whether it is disabled (and therefore ignoring Incoming Actions). This parameter can also be changed in the Translator List (shortcut SPACE).

8.2.3 Stop Processing

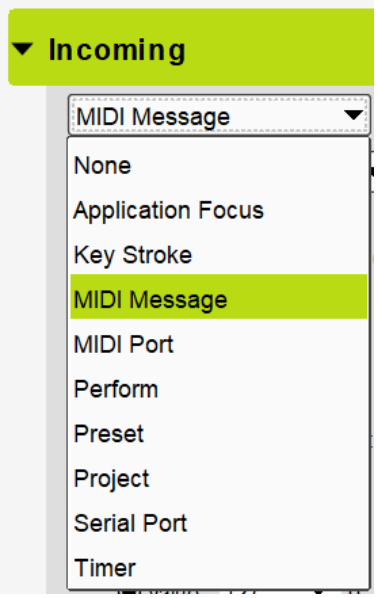
If this is enabled, successful completion of this translator's Outgoing Action will cause the rest of the translators in the current preset, and in following presets, to be ignored. This is useful for multiple-part presets that have different processes depending on different defined actions. In general, it is essential if you want to ensure that one incoming event is only processed by the first translator that matches.

Stop Processing is also useful for optimizing performance of very large projects with thousands of Translators (see chapter 13.7.2 Use "Stop Processing").

8.3 Incoming Actions (Trigger)

Incoming Actions define the triggers which Bome MIDI Translator can detect and act on. Bome MIDI Translator can recognize a range of different types of incoming actions.

See chapter 9, Actions, for more information using the individual action types.



8.4 Rules

The Rules represent a rudimentary scripting language for advanced usage. The Rules section is executed each time an incoming event matches the Incoming Action. The Rules allow you to process the incoming event parameters and apply logic and math to it. There is also global “memory” (i.e. global variables) that you can use in Rules.

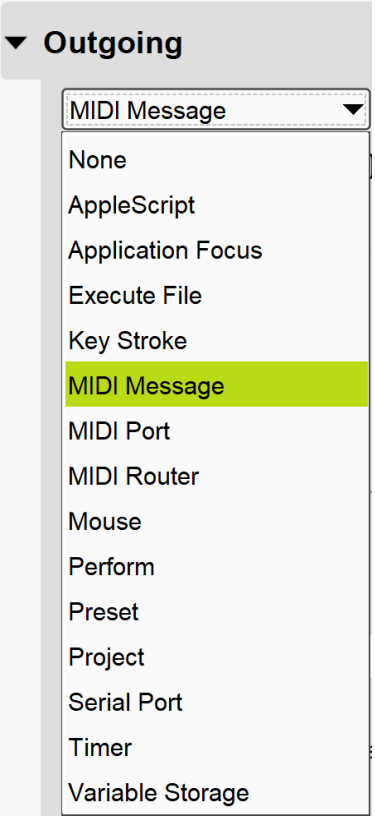
See the Rules and Variables chapter on page 97 for more information.

8.5 Outgoing Actions

Outgoing Actions are executed when the Incoming Action is triggered.

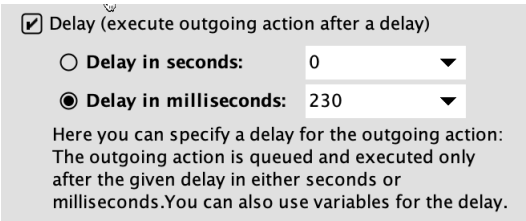
Bome MIDI Translator can output a range of different outgoing actions, as well as function with translators that are composed solely of rules with no defined outgoing action.

See chapter 9, Actions, for a description of the different action types.



8.5.1 Delaying Outgoing Actions

All Outgoing Actions can be delayed so that they will be executed after some seconds, or milliseconds.



Millisecond Precision

If you need millisecond precision, specify the delay in milliseconds. Note that 1000 milliseconds are equal to one second, so if you specify 2500 milliseconds the outgoing action will be executed after 2.5 seconds.

Using Variables for Delay

You can specify the delay with a variable. The amount to delay is evaluated when starting the delay, so even if the value of a variable changes while waiting for the delayed action to be executed, the delay will stay the same.

Using Global Variables in the delayed Outgoing Action

Of course you can use global variables in the Outgoing Action, but keep in mind that they are global and if they are changed while waiting for the delayed action, the delayed Outgoing Action will use the new value of the global variable.

Using Local Variables in the delayed Outgoing Action

Any delayed action can use local variables, they “stick” with the action.

Rules

If your Translator has rules, they are executed immediately, i.e. before the delay of the Outgoing Action starts “ticking”.

Cannot Cancel a Delayed Action

It is not possible to cancel a delayed action once it's triggered.

8.6 Editing Actions

The actions are edited in the right properties inspector. The Incoming and Outgoing areas have a drop down list where you you define the type for the given action. Depending on the type, an editor will be visible with more or less options.

All edits are generally immediately active so it is easy to test different variations. But be aware: **there is no Undo function!**

Whenever an action is not fully defined (e.g. data string is missing), or the action definition is not correct, an error message under the type selector will be displayed.

You can use the keyboard shortcut `Ctrl+6` / `Command+6` to quickly jump to the Incoming Action definition, and `Ctrl+8` / `Command+8` for the Outgoing Action.

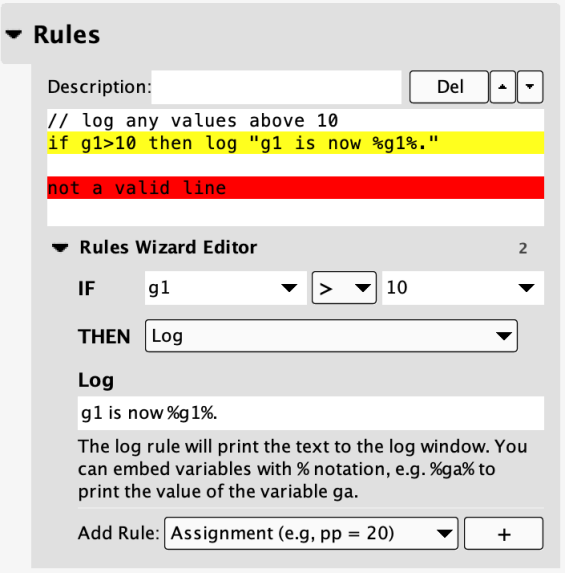
8.7 Editing Rules

The Rules editor has a free form text area, where you can see the rules like program text. You can either edit the text directly, or use the Rules Wizard Editor for point+click editing.

The current editor line is marked in yellow. Any lines that are not in the correct format are marked in red.

It is good practice to add comments to longer Rules sections so that you'll remember what the rules actually do. A comment line starts with `//`.

Use the keyboard shortcut `Ctrl+7` / `Command+7` to quickly jump to the rules section.



9 Actions

Actions are the core of the Translator entries: the Incoming Action defines on which event a Translator is triggered. The Outgoing Action defines what is executed if the Translator got triggered.

There are a number of different Action types in MIDI Translator. This chapter explains all of them.

9.1 MIDI Message Action

The MIDI Message action is used for reacting on incoming MIDI messages from MIDI devices or virtual MIDI ports, or for sending MIDI messages.

The MIDI Message Action is fully working in translation projects running on the BomeBox.

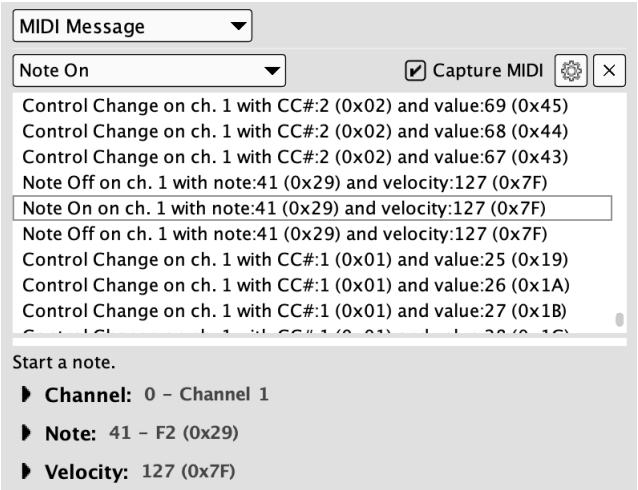
9.1.1 Incoming MIDI Message

With MIDI Translator, you can use any MIDI message as a trigger for your actions. You can either select from a number of predefined messages (simple mode), or use the raw/system exclusive mode to enter the MIDI message as raw hexadecimal values (raw mode).

Whenever a MIDI message is received, all translators with the incoming action type MIDI are checked in the order they appear in the preset(s). The incoming message definition is compared to the incoming action of each translator and if they match the Rules and Outgoing Action are executed.

Capture MIDI

The simplest way to define a new MIDI incoming action is to use the *Capture MIDI* feature (covered in the Quick Start guide) to capture the incoming MIDI information while you are pressing a MIDI keyboard key, turning a knob or otherwise. By default, all incoming messages will be displayed as an English description (simple mode). Only System Exclusive, and non-standard messages will be displayed in raw format using hexadecimal notation.

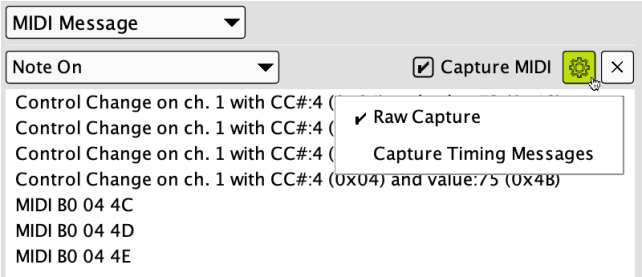


Clicking on a message in the capture panel will select that message as incoming trigger.

Uncheck *Capture MIDI* (or press Alt+C) to stop displaying incoming MIDI data. Click the X button to close the Capture MIDI panel.

> Note that *Capture MIDI* always displays incoming MIDI messages from all currently opened MIDI devices, regardless of selected MIDI ports for the translator.

Capture Options

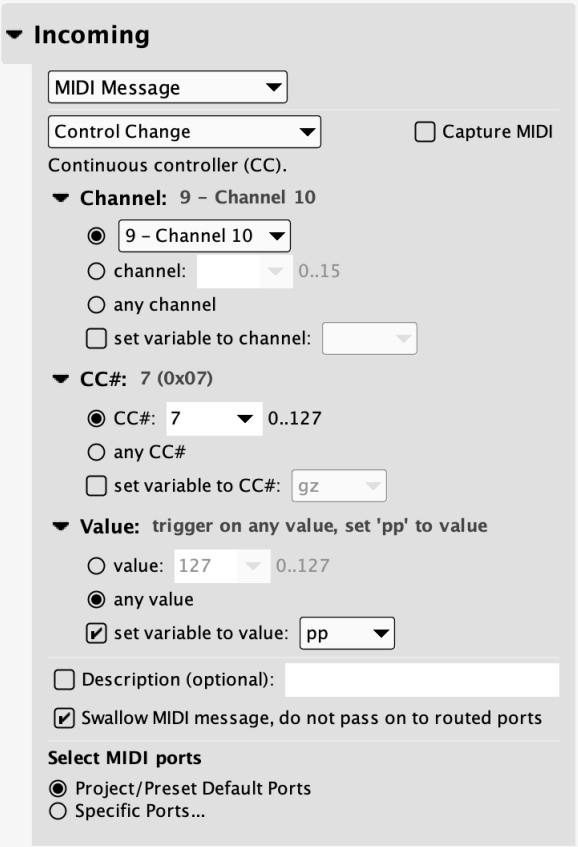


Use the gear icon next to the X icon in order to select how and which MIDI messages are captured:

Raw Capture: if checked, all captured messages are displayed in raw, hexadecimal format. By default Raw Capture is not checked, so that captured MIDI messages are displayed in simple format.

Capture Timing Messages: if checked, all timing messages are captured, too. By default this is off, so that regular timing messages won't clutter the capture box.

Editing MIDI Messages



You can now change the message as you like, e.g. to make it match any channel, or a specific channel set by a variable.

Variables are covered more completely in the Rules section. You can use local variables (defined as double-letter pairs such as 'pp' and 'xx' in the rules section to modify the incoming values to fit your needs. You can also use global variables (defined as two-letter variables beginning with the letters 'g' through 'z') to pass variables back and forth between translators.

Description of MIDI Message

This option allows you to write a short description of the Incoming Action for the translator that will show up in the main program interface. This can make complex translator setups much easier to work with and navigate.

Swallow MIDI Message / Do Not Route

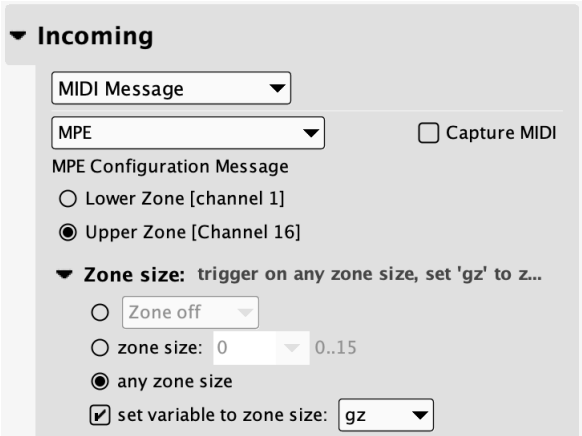
Use this option to prevent an incoming MIDI message to be routed to the MIDI Thru destinations specified in the Project Router. When an incoming MIDI message matches at least one Incoming Translator trigger, and the Rules will not skip the Outgoing Action, the message will not be sent to the MIDI Router. By specifying *None* as Outgoing Action, you can filter out MIDI messages from the MIDI Router.

Select MIDI Ports

The Incoming MIDI action can be assigned to listen on a specific port if so required. Select the *Specific Ports...* radio button, then check the MIDI port(s) you wish for the translator to listen to.

We recommend using *Preset Default MIDI Ports* where possible, instead of selecting the MIDI Ports on Translator level.

MPE



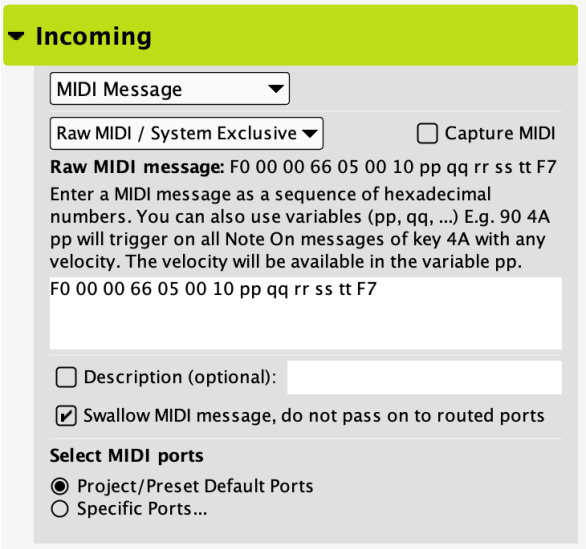
You can react to incoming MPE setup messages by selecting the MPE message type.

In the example at the right, the Translator is triggered for MPE Configuration Messages of the upper zone. When such Configuration message is received, the variable `gz` will be set to the zone size (of the upper zone). For example, if a MIDI device configures channels 14 and 15 as upper zone (zone size = 2, master zone channel = 16), the variable `gz` will be set to the value 2.

Raw MIDI / System Exclusive

When using the raw mode, you can enter any MIDI message, including system exclusive messages as a sequence of hexadecimal numbers. Check the MIDI Specification or online sources for more information on raw MIDI message definitions.

Also, invalid, partial, and multiple concatenated MIDI messages are possible.



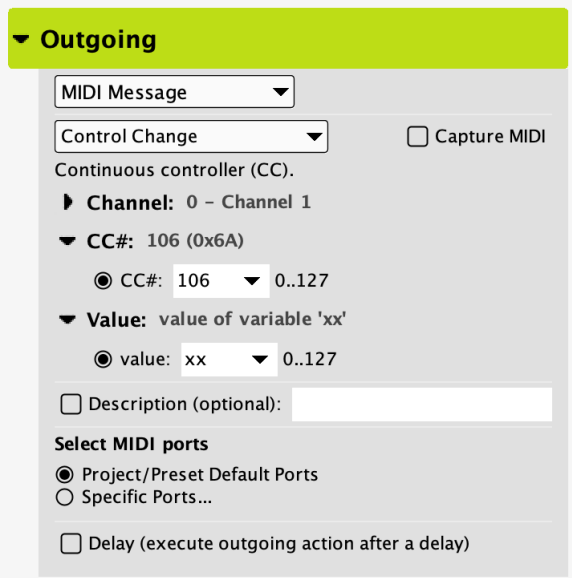
You can embed variables directly into the raw MIDI string instead of a number. In that case, the MIDI message will match any values at that position, setting the variable to the incoming MIDI message's value. Variables in the Incoming MIDI string will always be changed upon a matching incoming MIDI message, and not touched if the MIDI message does not match. We recommend using local variables here, as they are private to an incoming message, and don't collide if multiple matching MIDI messages arrive simultaneously.

Raw MIDI Examples

The following are some sample incoming raw MIDI string examples, along with description. For further instructions on using variables, go to the chapter Using Rules and Variables.

RAW MIDI STRING	TYPE	CHANNEL	CONTROLLER/ NOTE/PROGRAM	VALUE
`9F 6F pp`	Note On	16	111	ANY
`BA ww 7F`	Controller	11	ANY	127
`B4 xx pp`	Controller	5	ANY	ANY
`C4 nn`	Program Change	5	ANY	n/a
`pp qq rr`	any 3-byte MIDI message	any	any	any
`C2 10 B2 00 40`	Program Change followed by Control Change	3	Program #16 Controller #0	64
`F0 7F 7F 04 01 pp qq F7`	Master Volume (Universal System Exclusive)	n/a	`pp` receives LSB of master volume `qq` receives MSB	

9.1.2 Outgoing MIDI Message



When using MIDI as the Outgoing Action, the translator sends one or multiple defined MIDI messages when the Incoming Action is triggered.

In the screenshot to the right, the output value of the translator's outgoing MIDI message includes an `xx` variable, meaning that any number of rules could have been used to assign value to this variable depending on many factors.

Comparison with Incoming MIDI

Most options from the Incoming MIDI trigger are available in the Outgoing Action, too. The Swallow option is not available for outgoing actions, as it only makes sense for the Incoming Action. Also, you cannot set variables in the Outgoing Action's MIDI message , but you can send messages that take the values of variables which are already defined into account.

Capture MIDI

Capturing MIDI messages for Outgoing MIDI works the same as Incoming MIDI: the capture list displays all MIDI messages that are received from all currently open MIDI devices. In particular, the list does not display MIDI messages that are sent out to MIDI OUT ports. See the Log Window for monitoring MIDI OUT activity.

Selecting the Outgoing MIDI port

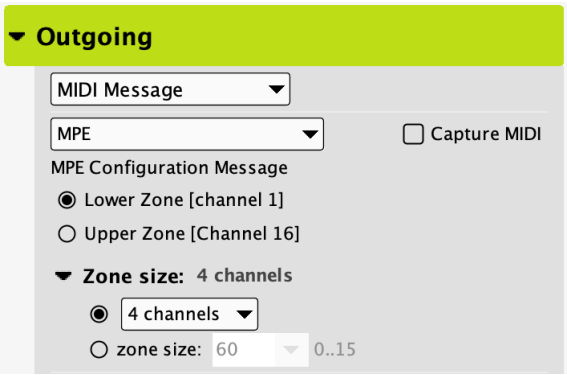
Outgoing MIDI actions are transmitted by default on the Preset Default MIDI ports, or if specified, on specific MIDI ports unique to the individual Outgoing MIDI action. For that, select *Specific Ports...* on the radio button selector and check each MIDI OUT port you wish to use.

Description

Also, an optional Description can be entered for each Outgoing Action that will give a plain text description that can be viewed from the program main interface.

MPE

In the MIDI Outgoing action, select MPE from the list for a convenient way to output the MPE Configuration message.



You can either send the MPE Configuration for the lower zone or for the upper zone. For each zone, you can specify the size of the zone (in channels), or turn off the respective zone (0 channels). You can also use a variable to define the zone size. The variable value should be in the range of 0 to 15.

In the screenshot above, the outgoing action sends the MPE Configuration Message to set up the lower zone with 4 channels (channels 2...5). Channel 1 is the zone master channel.

Raw MIDI / System Exclusive

Similar to the Incoming MIDI trigger, you can specify the outgoing MIDI message as a raw MIDI message in form of a string of hexadecimal numbers. Check the MIDI Specification or online sources for more information on raw MIDI message definitions.

You can include variables in the outgoing raw MIDI string. In that case, before sending the message to the MIDI port, the variable is replaced with its value and that replaced string is sent. The variable's value is not changed.

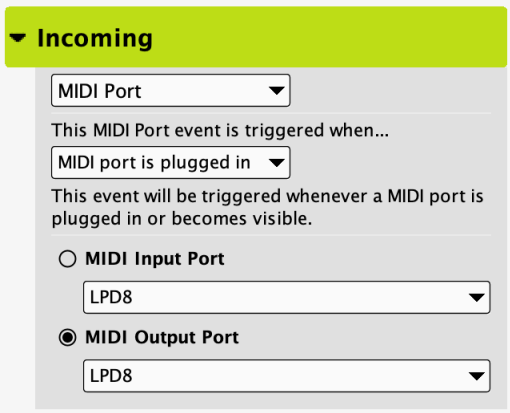
9.2 MIDI Port Action

The MIDI Port action allows triggering actions when a MIDI device gets plugged in or is unplugged, or to re-assign MIDI Port Aliases dynamically from within your translations.

The MIDI Port Action is fully working in translation projects running on the BomeBox.

The MIDI Port action is available since version 1.9.

9.2.1 Incoming MIDI Port Action



The incoming MIDI Port action is triggered when an event regarding a physical or a virtual MIDI port occurred.

You can select the MIDI INPUT or the MIDI OUTPUT device on which this action is defined. You can also select a MIDI Port Alias here: in this case the action is triggered when the assigned device is plugged in or out.

There are 5 different types of the incoming MIDI Port action.

MIDI Port is Plugged In

This action is triggered when the given device is plugged in, or the respective virtual MIDI port is created.

MIDI Port is Unplugged

This action is triggered when the given device is unplugged, or the respective virtual MIDI port is removed.

MIDI Port is Opened

This action is triggered when the given device is opened within MIDI Translator Pro. This is the case when the MIDI port exists and when it is used (somewhere) in the project. This happens when a project device is plugged in, or a MIDI route is added or enabled, or an alias is assigned so that it becomes a project port.

A *pending* port (used in the project, but not plugged in) will not trigger the MIDI Port is Opened action.

MIDI Port Got Closed

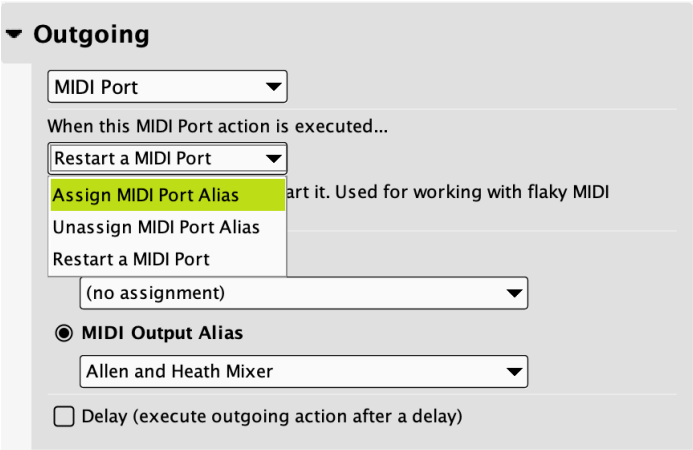
This action is triggered when a previously open device is either not used in the project anymore (e.g. because of redefining a route or unassigning the alias), or because it is unplugged.

MIDI Port Open Error

This action is triggered when a MIDI port is supposed to be used in the project, and it is plugged in, but MIDI Translator failed opening it. The main reason why this could happen is on Windows, when a MIDI INPUT port is already used by another software.

9.2.2 Outgoing MIDI Port Action

With the MIDI Port Outgoing Action, you can assign or unassign MIDI Port aliases, or restart a MIDI Port.



Assign MIDI Port Alias

When this Outgoing Action is invoked, the given MIDI Port Alias is re-assigned to a real MIDI device. Like this, you can dynamically change the MIDI Port to be used in your presets and translators, when they operate on the given MIDI Port Alias.

Unassign MIDI Port Alias

When this Outgoing Action is invoked, the given MIDI Port Alias is assigned to None, i.e. the alias is not pointing to an existing MIDI device anymore. This can be useful in response to a MIDI Port Unplugged incoming action, or when you want to release the resources taken by the given MIDI Port.

Restart a MIDI Port

When this type of the MIDI Port action is invoked, the given MIDI Port is temporarily closed, and re-opened after a few seconds. This should not be necessary under normal circumstances, but we have had reports from broken devices or drivers, which could be reanimated again by closing and opening the port.

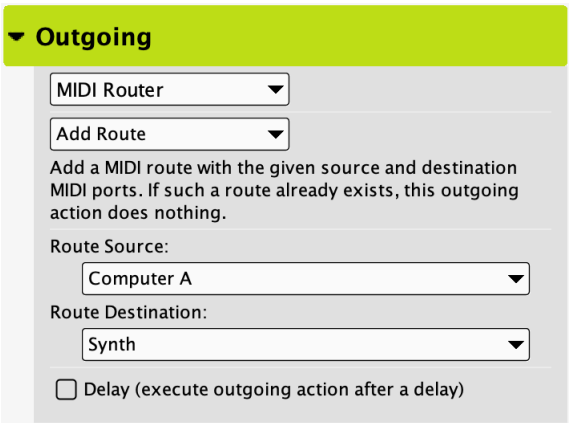
9.3 MIDI Router Action (Outgoing)

With the MIDI Router action, you can dynamically add or remove MIDI Routes, and enable or disable existing routes. For more information on MIDI Routes see the MIDI Router chapter on page 30.

The MIDI Router Action is fully working in translation projects running on the BomeBox.

The MIDI Router action was introduced in MIDI Translator Pro version 1.9.

9.3.1 Outgoing MIDI Router Action



The MIDI Router action allows adding, removing, enabling, or disabling MIDI Routes. A MIDI Route is defined as a source MIDI Port and a destination MIDI Port. For both the source and the destination, you can also use MIDI Port Aliases.

Add Route

This type of the MIDI Router action will, when invoked, add the given route. In the screenshot above, the action will create a route from the MIDI INPUT `Computer A` to the MIDI OUTPUT `Synth` is created. If the corresponding route already exists at time of invocation, the action will not do anything.

When adding a route, the source and the destination ports will become Project MIDI Ports (if they are not already).

Remove Route

This type of the MIDI Router action will, when invoked, remove the given route. If the corresponding route does not exist at time of invocation, the action will not do anything.

Enable Route

This type of the MIDI Router action will, when invoked, enable the given route. If the corresponding route is already enabled, or the given route does not exist, the action will not do anything.

Disable Route

This type of the MIDI Router action will, when invoked, disable the given route. If the corresponding route is already disabled, or the given route does not exist, the action will not do anything.

9.4 Keystroke Action

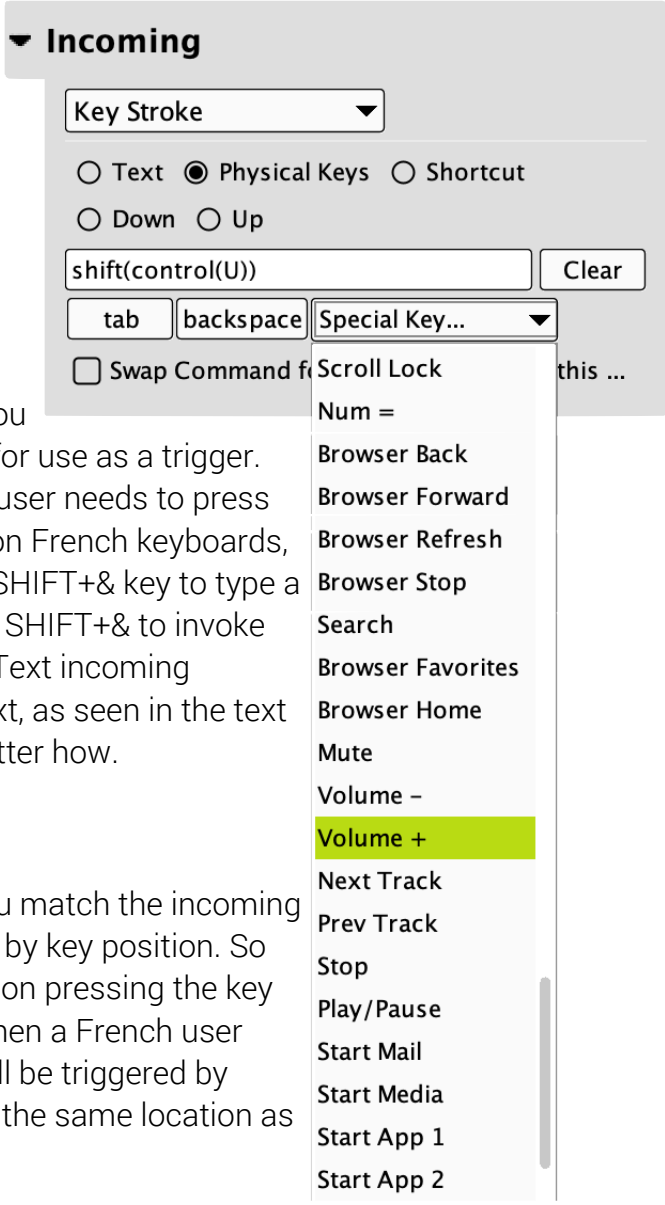
Keystroke actions react on keys pressed on your (computer) keyboard, or emulate such keys as if they were pressed.

The Keystroke Action only works for Incoming Actions in translation projects running on the BomeBox.

9.4.1 Incoming Keystroke

If you want to trigger a translator by pressing (or releasing) keys on your computer keyboard, use Keystroke as Incoming Action.

Keystroke-based incoming translator actions can be defined in different ways.



Text

The *Text* incoming keystroke type lets you define a text (or just a letter) to be typed for use as a trigger. For example, if you enter the text “1”, the user needs to press the 1 key to trigger the action. However, on French keyboards, there is no key for “1”, you need to press SHIFT+& key to type a “1”. Thus, French users will need to press SHIFT+& to invoke the same action. As a rule of thumb, the Text incoming keystroke action is triggered when the text, as seen in the text field, is produced by the keyboard, no matter how.

Physical Keys

The *Physical Keys* keystroke type lets you match the incoming action by physical keys. They are defined by key position. So entering a “1” key here will always trigger on pressing the key which produces the “1”. Consequently, when a French user opens the same project file, the action will be triggered by pressing the “&” key because the key has the same location as the “1” key on an English keyboard.

Shortcut

The *Shortcut* keystroke type is for typical shortcuts consisting of one or more modifier keys (like Shift, Control, ...) with one or more letters or other keys. Use the checkboxes to selectively create shortcuts which are difficult or impossible to enter directly, like

Command/Windows-Tab. On macOS, you can display shortcut modifiers as icons or as key names. See the Appearance section in the Settings. The shortcut type is available since MIDI Translator Pro version 1.9.

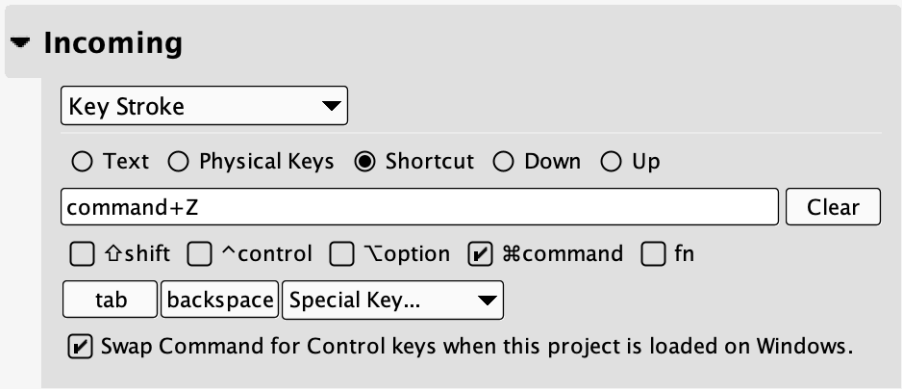
Down/Up

The *Down* and *Up* types let you trigger the action on only pressing down or releasing the specified key. Only one single physical key is possible here. It is often used to capture the Down/Up state of modifier keys such as SHIFT, CONTROL, etc.

Key Sequences

For the *Text* and *Physical Keys* types, you can specify arbitrary sequences and key combinations to serve as a trigger. Parenthesis are used to mark simultaneous presses of keys. This can be conveniently used to trigger on keyboard shortcuts. For example, `Shift(Ctrl(A))` will only trigger if you press A with the `Control` and `Shift` keys together. Similarly, you can also invent new combinations like `A(B(C))` which will only trigger if you press `C` while holding down `A` and `B`. Last, but not least, you can also create key sequences which must be fully typed to trigger the action. For example, you the sequence `A B Ctrl(D A)` will only trigger if you press `A` (down and up), then `B`, then `Ctrl` Down, followed by `D` and `A`, and then `Ctrl` Up.

Swap Command for Control keys

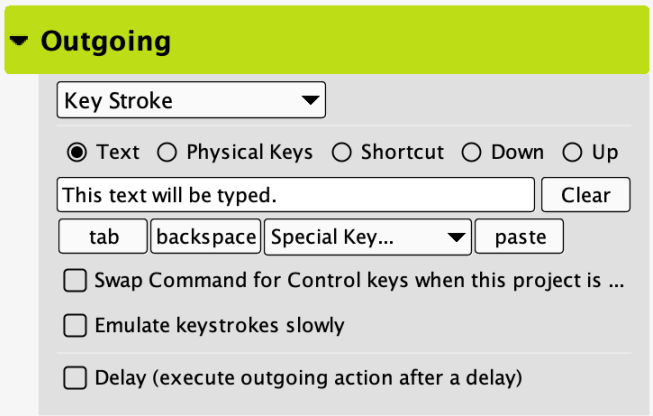


The *Swap Command key for Control* option is very useful when defining shortcuts: many standard shortcuts use the Command key on macOS (e.g. Command-Z for Undo), but on Windows, the same shortcut is used with the Control key (Ctrl-Z). If you check this option, a shortcut defined with Command on a Mac computer will appear as the same shortcut with Control when the same project file is loaded on Windows. And vice versa when created on Windows. Therefore, this option is very useful when the same project file is to be used on Windows and on macOS.

When **entering** key strokes, use the BACKSPACE key to remove the last entered key, and the TAB key to remove focus from the keystroke field. To enter a TAB or BACKSPACE key as trigger text, use the respective buttons below the keystroke field. Use the **Clear button** to entirely remove all entered keys and start over. The Special Key drop-down list allows you to enter a special key which might not be present on your keyboard.

On **macOS 10.9** and later, you need to enable MIDI Translator Pro in System Preferences, Security & Privacy → Privacy → Accessibility.
For macOS earlier than 10.9, go to System Preferences, Universal Access: check 'access for assistive devices'.

9.4.2 Outgoing Keystroke



For emulating keystroke presses (and releases), and to emulate typing entire sequences of keys, use the Outgoing Keystroke action.

Incoming actions can be translated to any combination of keystrokes, for use in complex macro routines to control 3rd party program functions. The best place to start with programming keyboard combination macros is to consult your software's instruction manual and determine what keyboard shortcuts are available to you.

Keyboard Emulation outgoing actions can either be typed text, physical key press/key combination events, or individual Key Up / Key Down events.

When **entering key strokes**, use the BACKSPACE key to remove the last entered key, and the TAB key to remove focus from the keystroke field. To enter a TAB or BACKSPACE key into the keystroke field, use the respective buttons below the keystroke field. Use the Clear button to entirely remove all entered keys and start over. The **Special Key** drop-down list allows you to enter a special key like Mute, Browser Back, etc. which might not be present on your keyboard.

The emulated keystrokes are always sent to the currently active application. When MIDI Translator is the active application, keystrokes are not sent. You can use the incoming Application Focus action to disable a preset if the correct application is not focused. Vice versa, you can use the Outgoing Action Application Focus to focus the correct application prior to emulating the keystroke.

Text

The Text mode (new in version 1.8) allows emulating keystrokes so that they type the given text (i.e. characters and not keys). This is particularly useful for entering numbers, special characters, and international letters which may have different positions or ways to enter on different keyboards.

In the text field, enter the text to be typed to define it for this Outgoing Action.

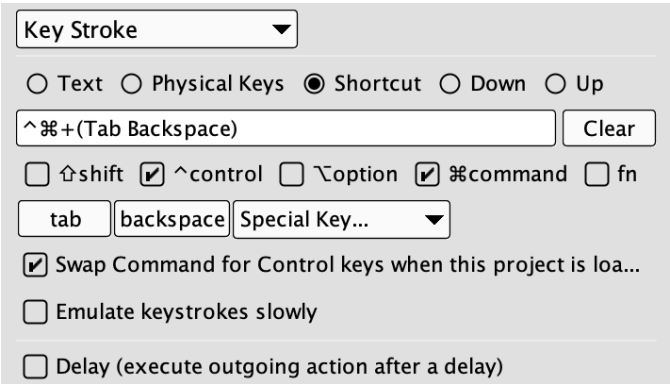
When executing a Text string, MIDI Translator figures out the necessary key strokes for each letter, emulating SHIFT and other modifier keys as necessary. For example, to emulate typing the square bracket "[", MIDI Translator will just emulate typing that key on an English keyboard, while it will emulate pressing RightAlt+8 on a German keyboard.

Physical Keys

Emulating physical keys is similar to Text, but it will always execute the same keys (and not characters), no matter which keyboard or language is selected. Shift and other modifiers are treated as individual keystrokes. So, for example, when you create an outgoing physical keystroke definition `Ctrl([)` on an English keyboard, it will produce `Ctrl+[` on your computer, but `Ctrl+ü` when MIDI Translator is running on a computer with German keyboard selected.

Shortcut

In this mode, you can easily define shortcuts like `Shift-Ctrl-Y` or `Command-Tab`. The checkboxes can be used to construct your own shortcuts which are difficult to type directly into the field. For more information, see the description of the Shortcut mode of the Incoming Keystroke Action.



Key Down / Up

You can emulate pressing just one key down (press) or up (release). This is useful when you want to keep modifier keys like Shift pressed for subsequent Outgoing Actions (e.g. combine with Mouse Outgoing Action).

For Keystroke Down, you can optionally define a repeat delay, so that the key is repeated automatically as long as you don't send the corresponding Key Up action.

Example: MIDI Note to Keystroke with Repeat

If you want to execute a key event, and have the key repeated as long as you press a key on your MIDI Keyboard, you will need two translators:

1. The first translator has MIDI as incoming trigger. Use MIDI Capture, press the MIDI key and select the Note On variant in from the captured events. Make sure you select “any” for the velocity. As outgoing action, use “Key Stroke Down”. Enter the letter/key to be “typed” in the text field (e.g. "X"). Enable the Key Repeat box.
2. Now duplicate the first translator and edit it: change the incoming message to Note Off (keep everything else). Change the Outgoing action's Key Stroke to “Up” and re-type the letter (e.g. “X”).

Now switch to a text editor and press the key on your MIDI Keyboard down. As long as you keep it pressed, the letter "X" will be typed repeatedly. Once you release the key, the keystroke emulation stops, too.

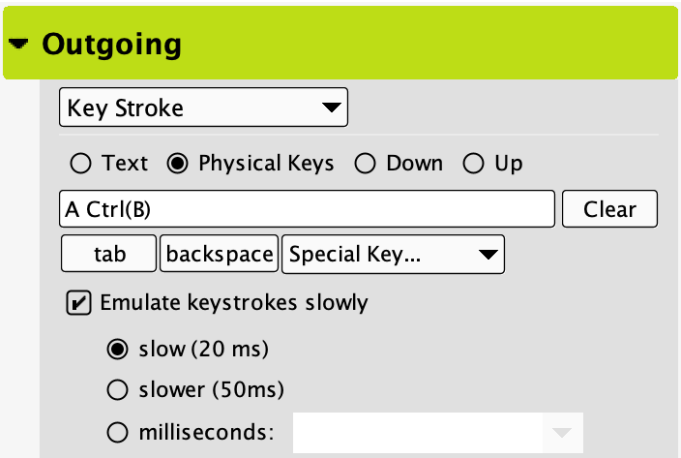
Swap Command for Control keys

See Swap Command for Control keys in the incoming keystroke action.

Emulate Keystrokes Slowly

Normally, keystrokes or keystroke sequences are emulated as fast as possible. Some target applications, however, require keys to be (virtually) typed a little slower in order to be fully processed. In this case, enable “Emulate keystrokes slowly” (new in version 1.9).

The “slow” setting will delay every subsequent keystroke by 20ms, which only introduces a slight delay in between keystrokes. For many programs, the 20ms inter-key delay is enough to fully process all keys.



If the target program is executing lengthy operations (such as “save”) during processing of the keystroke sequence, you can try with the “slower” keystroke delay setting, which will delay every keystroke by 50ms.

For longer inter-key delays, use the third option, where you can specify the delay in milliseconds directly. You can also use a variable here for the delay. The variable will be evaluated when the first keystroke is triggered, all subsequent keys of the keystroke sequence are executed with the same inter-key delay, even if the variable changes its value during processing.

Note that in slow keystroke emulation, *every* raw keystroke is affected by the delay: also modifier keys like Shift and Control are delayed, and pressing and releasing keys are also delayed appropriately.

Example

The following table lists the emulated keystrokes when using slow emulation.

Time (ms)	Action
0	Trigger *Keystroke Sequence: "A Ctrl(B)", with delay 20ms*
0	Execute "A Down"
20	Execute "A Up"
40	Execute "Ctrl Down"
60	Execute "B Down"
70	Execute "B Up"
100	Execute "Ctrl Up"

Scheduled Keystrokes

When *Emulate Keystrokes Slowly* is enabled, all such keystrokes are scheduled, also with respect to other delayed keystrokes. Like that, it is guaranteed that keystroke sequences are always executed in order, and newly triggered keystroke sequences are emulated one after another.

In contrast, keystrokes without "emulate keystrokes slowly" will always be emulated as soon as possible. Therefore, we recommend that you don't mix slow and direct keystrokes in one translation project.

> It is recommended that you either enable "slow" keystrokes for *all* emulated keystrokes in a project, or for *none*.

Example

The following table lists the keystroke emulation sequence when four translators trigger an outgoing keystroke or keystroke sequence. One of them is not delayed, and will interfere with the enqueued keystrokes.

Time (ms)	Action
0	Trigger *Keystroke Sequence: "Ctrl(W)", with delay 20ms*
0	Execute "Ctrl Down"

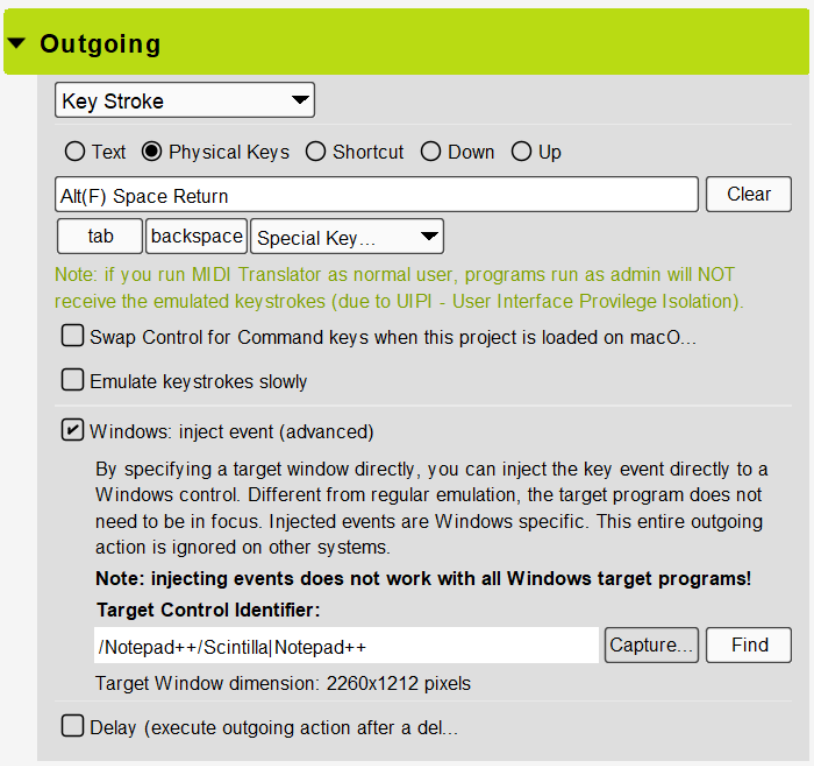
20	Execute “W Down”
23	Trigger Keystroke Sequence: “Shift(Y)”, delay 50ms
40	Execute “W Up”
60	Execute “Ctrl Up”
80	Execute “Shift Down”
130	Execute “Y Down”
180	Execute “Y Up”
185	Trigger Keystroke Down “S”, delay 50ms
200	Trigger Keystroke Sequence “A”, no delay
200	Execute “A Down” → interfering with scheduled keystrokes
200	Execute “A Up” → interfering with scheduled keystrokes
230	Execute “Shift Up”
280	Execute “S Down”

9.4.3 Injected Keystroke Events (Windows only)

On Windows, you can inject keystroke events directly to a specific application. For programs with which it works, you can type into programs without the need to activate them.

Injecting key strokes only works on Windows. It does not work with every program, however, and you will need to test with your application of choice.

Injecting keystrokes can make your projects much more robust because it will still work when (accidentally) another program is activated, and you can control multiple programs with key strokes without the need to switch the active window.



For injecting key strokes, check that option in the Outgoing action, then click on the Capture button. Moving your mouse will now show a green rectangle around every recognized window and sub window. Click on the window or sub window you want to target with the injected key strokes. Use the Find button to see if the window definition is sufficient to find the corresponding window.

We recommend that you always test if the injected key stroke works. Often, application windows are layered and you may need to experiment a bit to find the suitable sub window for injecting key strokes.

Sometimes, it can even work to directly inject, say, the space bar to a button sub window, effectively clicking the button.

9.5 Timer Action

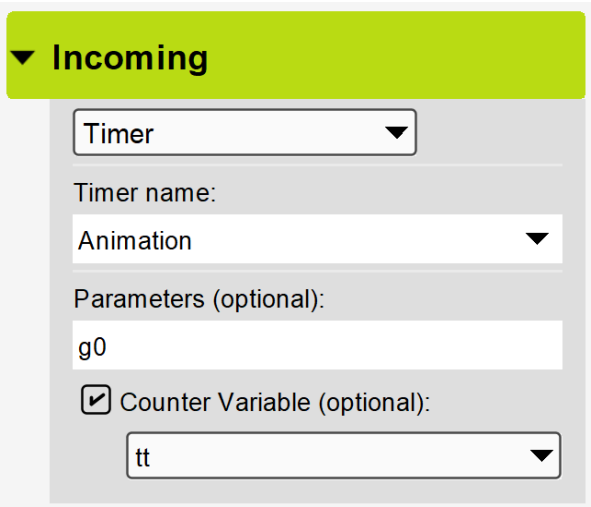
With a Timer, you can create delayed and recurring actions. A Timer will, when its time has elapsed, feed an Incoming Event into the processing engine. You can execute a Timer with an Incoming Action just like any other event in a translation project.

In general, this is how a timer works when activated in an Outgoing Action:

1. Wait for the duration of the *Initial Duration* (one-shot timers only have a *Duration*).
2. Fire off an Incoming Timer Event with the name of the Timer into the MT processing engine.
3. In the engine, all translators with Incoming Timer action where the name matches the event's timer name, will be executed.
4. If this is a *Once* Timer Action, the timer will be stopped.
5. If this is a *Multiple Times* timer and number of executions has reached the *Repeat Count*, stop timer.
6. Otherwise, wait for the *Repeat Duration*.
7. Start over at 2.

The Timer Action is fully working in translation projects running on the BomeBox.

9.5.1 Incoming Timer



Incoming Timer actions are events that will trigger once or multiple times depending on the Timer settings in the respective Outgoing Timer Action. Incoming Timer actions must be tied to an already-existing Outgoing Timer Action in order to work properly.

Defining Incoming Timer actions is very simple — the main option is the selection of which Timer to use: the name.

Note that you can specify multiple translators with the same Incoming Timer as incoming action. That's a convenient way to execute multiple outgoing actions with one single trigger.

Parameters

Optionally, you can pass values along with the timer event. These are specified in the Outgoing Timer Action as parameters. Here, in the Incoming Trigger, you can get the values of the parameters in the Parameters list. The first variable you specify will receive the value of the first parameter value specified in the Outgoing Timer, the second parameter from the second value, and so on. You can use up to 10 parameters from the Outgoing Timer. If you specify more variables here than the Outgoing Timer event has parameters, the extra variables will not be changed. Conversely, it's also OK to specify fewer variables here than the Outgoing Timer sends parameters: the additional parameters will be ignored in the processing of this Translator.

In the screenshot above, the global variable `g0` is set to the value of the first parameter specified in the respective outgoing Timer Action.

This mechanism works in the same way as passing parameters using the Perform Action.

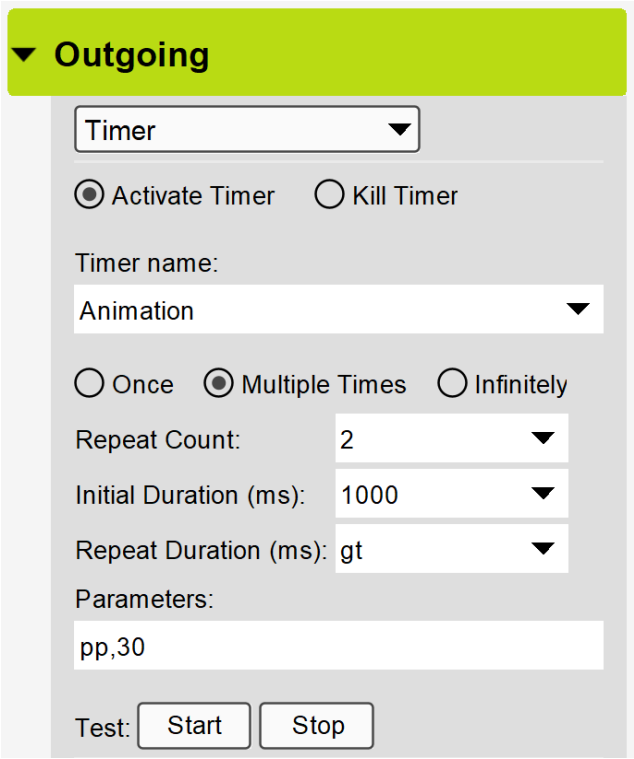
Timer Parameters are available since MIDI Translator Pro 1.9.2, and BomeBox firmware 1.6.

Counter Variable

Optionally, you can define a variable that receives the current repeat count. On first invocation of this timer, the counter is 0. With every subsequent Timer event, the counter is increased by one. If the Outgoing Timer action is executed again from an Outgoing Timer Action, the counter will reset to 0 and start incrementing with every invocation.

The Timer Counter feature is available since MIDI Translator Pro 1.9.2, and BomeBox firmware 1.6.

9.5.2 Outgoing Timer



There are two main types of Outgoing Timer actions — Activate Timer and Kill Timer. Timers can be instantiated by selecting them as an Outgoing Action, and setting their appropriate repeat times and other options. Timers are usually associated with other translators that have the Timer name as their Incoming Action.

For instance, if you want to repeat the Up arrow key as long as a condition is met, create an Outgoing Timer Action that sets the repeat rate, and then create a new translator which will output the keyboard emulation for the Up arrow key — using your existing Timer Translator as the Incoming Action.

Timer options include repeat occurrence (once, multiple times, indefinitely), Initial Duration, Repeat Duration, parameters, and testing functionality.

Timers are always Retriggering

If you start a Timer that is already running, the wait time will be refreshed with the new (initial) Duration, and the repeat counter will be reset to 0.

Activate Timer

When executing the Activate Timer Outgoing Action, a Timer event is fed into the processing engine. The event will trigger one or more times the respective Incoming Timer Action(s) with the same name as this timer.

If a timer of the given name is already running when Activate Timer is executed, the new timer replaces the previous running one.

Kill Timer

Executing a Kill Timer Outgoing Action will stop a running Timer of the given name. If no such Timer currently exists at time of execution, this Outgoing Action will not do anything.

The Kill Timer Outgoing Action can be used to prematurely stop a Multiple Timer before it finished all repetitions. And it is the only way to stop an Infinite timer.

Timer Name

The name you specify here is the identifier that serves to match the Incoming Timer action with the Outgoing Timer. When executed, the Outgoing Timer Action feeds a Timer event into the processing engine with this name. Subsequently, all Incoming Timer Actions with that given name will be triggered.

Once / Multiple Times / Infinitely

Here you select how often the timer is triggering the respective Incoming Timer Action. With *Once*, the timer is will be called exactly one time.

With *Multiple Times*, the field *Repeat Count* determines how often the Incoming Timer Action will be called. You can specify a number (e.g. 2, as in the example above), or use a variable to set the number of repetitions. You can stop a multiple timer before all repetitions are over by executing a Kill Timer outgoing action.

Last, but not least, specifying an *Infinite* Timer will cause the timer to be repeated forever. It can only be stopped via the Outgoing Kill Timer Action.

(Initial) Duration

Here, you specify the time it takes from executing this Outgoing Timer Action until the first Incoming Timer is triggered. The value is specified in milliseconds. Sometimes, it makes sense to specify 0 here, so that the timer is executed directly before the repeated (delayed) timer events are triggered.

For a Once timer, this is just the duration for the one and only invocation of the Timer.

Repeat Duration

For multiple and infinite Timers, you can specify the *Repeat Duration*, which is the time in between Timer invocations after the first one. Just like the initial duration, the Repeat Duration is specified in milliseconds.

Parameters

Since MIDI Translator Pro, version 1.9.2, and BomeBox firmware 1.6, you can pass up to 10 parameters to the Incoming Action. Specify them using a comma-separated list of variables or numbers in the Parameters field. When the Outgoing Timer Action is executed, the values of all parameters are evaluated and stored as numbers with the Timer event. Only the numbers are passed to the Incoming Timer Action. This works in the same way as parameters in the Perform Action.

Note that for repeating timers, the value of variables is only evaluated once when the Timer is initially started. If you use a global variable in the Parameters, and the variable changes value between invocations of the Timer, it will still always report the initial value to the Incoming Timer actions.

One-shot 0ms Timer

Popular with older versions of MIDI Translator Pro was a trick to use a one-shot timer with 0 duration: this will cause the current input event to be fully processed, and the timer event will be processed immediately, too — possibly already in parallel to the current event.

In current versions of MIDI Translator Pro, **the Perform Action** (see chapter 9.6 Perform Action) achieves the same in a more elegant and safer way. See also chapter 13.5 Leverage the Perform Action and 13.6 Multiple Actions in one Translator.

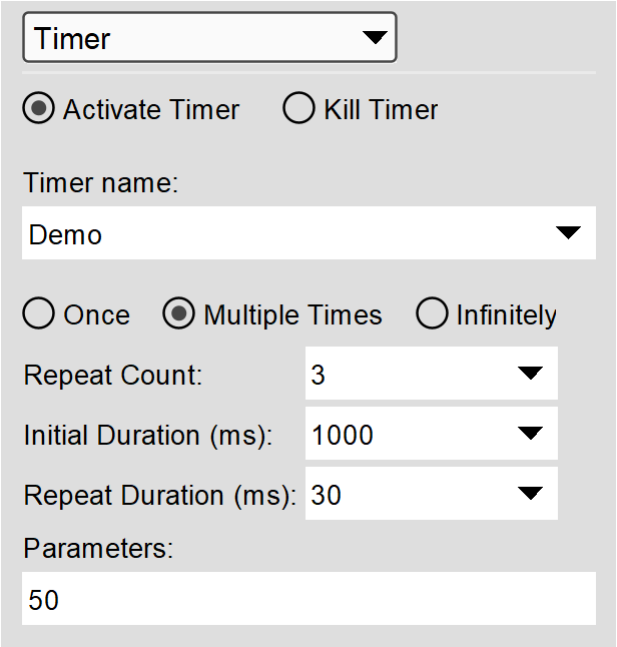
Timer Initial Duration vs. Outgoing Action delay

Note that for starting a timer, you can specify an Initial Duration, and there is also the possibility to specify an action delay (see Delaying Outgoing Actions). Those two delays add up, but there is a big difference: by delaying the outgoing action, the timer will not exist before the outgoing action delay has passed. So during that time you cannot kill or override that timer! Consequently, as a rule of thumb, you should avoid delaying an outgoing timer action, and rather use the timer's (initial) duration.

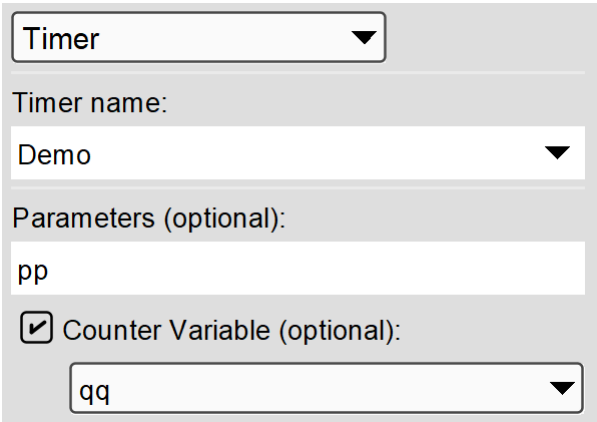
9.5.3 Example for a Timer

Consider the following Timer definitions:

Outgoing Timer:



Incoming Timer:



Time Line:

Time (ms)	Action
0	Trigger the Outgoing Timer Action above. The “Demo” timer event is inserted into the processing engine.
1000	The first invocation of the timer “Demo”, after the Initial Duration. The Incoming Timer Action above is triggered, setting `pp` to 50 (which was passed as first parameter), and setting `qq` to 0 (it’s the first invocation). As for all timer events, any other Translators with incoming type Timer and with Timer name “Demo” are triggered, too.
1030	The second invocation of the timer “Demo”, after the Repeat Duration has passed. Again, `pp` is set to 50. The variable `qq` is set to 1, because it’s the second invocation of the timer.
1060	The third and last invocation of the timer “Demo”, after the Repeat Duration has passed yet again. As before, `pp` is set to 50. And now the counter variable `qq` is set to 2, because it’s the third invocation of the timer.

9.6 Perform Action

The Perform Action is another way to feed back events and state in the engine. It is available in MIDI Translator Pro since version 1.9.1. The Perform Action is fully working in translation projects running on the BomeBox since firmware v1.5.3.

In general, it works like this: whenever an Outgoing Perform Action is executed, it feeds a named event into the translation engine. All Translators with a corresponding Incoming Perform Trigger will be triggered. Any parameters passed in the Outgoing Action will be relayed to the Incoming trigger.

The Perform Action is similar to the Timer Action, except that the corresponding Incoming Perform Trigger is triggered immediately. Also, an important difference is that Perform Actions will never be *retriggered*: every time an Outgoing Perform Action is executed, it will result in exactly one Incoming event into the translation engine.

This is how the Perform Action works when executed in an Outgoing Action:

1. Fire off an Incoming Perform event into the MT processing engine with the name of the Perform Action and any optionally given parameters.
2. In the engine, all translators with Incoming Perform Trigger and the Perform's name will be triggered. The optional parameter values are passed to every Incoming Action.

The Perform Action:

1. Is an elegant way to trigger multiple actions at once. See 13.6 Multiple Actions in one Translator.
2. Allows you to do a lot more with fewer global variables (because it passes values).
3. Allows you to set up a Translator that can perform multiple functions based on the parameters passed.
4. Allows you to create translation projects that are easier to maintain.

See also chapter 13.5 Leverage the Perform Action and 13.6 Multiple Actions in one Translator.

Parameters are passed by value

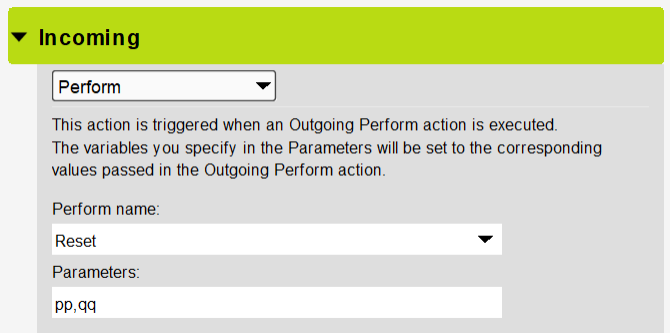
The parameters in the Outgoing Perform Action can contain variables. They are evaluated at the time of executing the respective Outgoing Perform Action. The resulting Perform Event that is fed into the MIDI Translation Engine contains only numbers as parameters – every variable is replaced by the value at time of executing that Outgoing Perform Action.

Beware of Infinite Loops

Always make sure that you don't create circular Perform Actions. If an Outgoing Perform Action triggers a Translator Entry which, in turn, will again execute an Outgoing Action with the same Perform name, it will result in an infinite loop. The MIDI Translator engine does not detect this and will be stuck in an infinite loop.

9.6.1 Incoming Perform Trigger

Incoming Perform Triggers are triggered by corresponding Outgoing Perform Actions with the same name. In the screenshot at right, the Incoming Perform Trigger is triggered, whenever an Outgoing Action of type Perform and with name "Reset" is executed.



Defining Incoming Perform triggers is very simple — the main option is the selection of which Perform to use (the name).

Additionally, you can specify one or more variables to receive the Perform parameters. At time of triggering an Incoming Perform, the given variables are set to the values passed in the executed Outgoing Perform action.

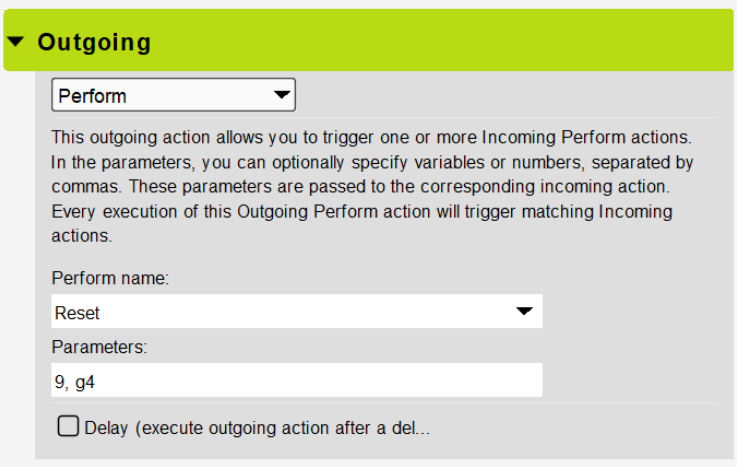
For example, if the Outgoing Perform Action triggered the "Reset" Perform with parameters "9, 100", then the example Incoming Perform action above is triggered and the local variable pp is set to 9, and `qq` is set to 100.

If the Outgoing Perform action specified fewer parameters than the Incoming Trigger, any surplus parameters in the Incoming Perform are set to 0.

If the Outgoing Perform action specified more parameters than the Incoming Perform, the additional parameters are ignored.

Note that you can specify multiple translators with the same Incoming Perform as incoming trigger. That is a powerful way to execute multiple outgoing actions with one single trigger.

9.6.2 Outgoing Perform Action



The Outgoing Perform Action serves to inject Perform Events into the translation engine. Every invocation of a Perform Event can optionally carry one or more parameters.

In the screenshot to the right, a Perform Action with name “Reset” and two parameters is triggered. The first parameter is number 9, and the second parameter is the value of the variable g4 at time of execution of the Outgoing Perform action.

The variables in the parameters are evaluated when the action is executed. So, even if g4's value changes by the time the Incoming Perform is triggered, it will receive the old value of g4. That's why you can also use local variables in the parameters: its value is passed, not the variable itself.

You can pass up to 10 parameters.

9.6.3 Trigger Perform via Command Line

You can trigger a named Perform via the command line using the following format:

```
/triggerPerform <name>,param1,param2,...
```

For example, to simply trigger the “Do Something” Perform, use it like this:

```
/triggerPerform "Do Something"
```

Or, to trigger the “Reset” Perform from the screenshots above, use this:

```
/triggerPerform "Reset,23,g2"
```

Note: you cannot use local variables as parameters when triggering Perform via command line.

9.6.4 Trigger Perform from the Rules

You can also trigger a named Perform action with a specific Rule. See 10.2.5 Perform for more information.

9.7 Preset Action

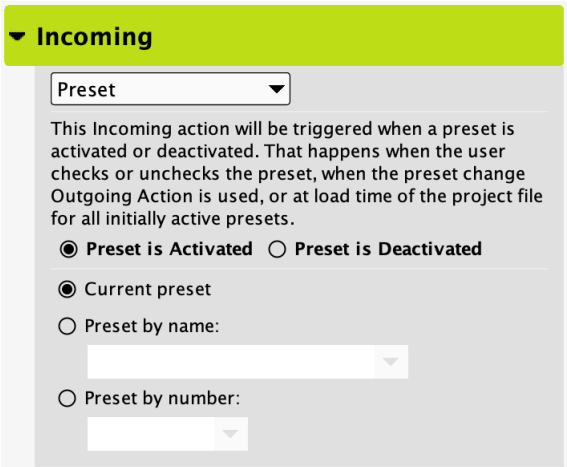
The Preset Action is used to dynamically manipulate translation presets from within your project, and to react on changes.

The Preset Action is fully working in translation projects running on the BomeBox.

> Note: this action was named “Preset Change” in older versions of MIDI Translator Pro.

9.7.1 Incoming Preset Actions

Incoming Preset Actions are activated when the preset active/inactive state is changed through some means. Preset Actions are useful for "one-off" type of actions that only occur once at the very beginning of the preset change. These actions are often 'reset' actions that could either redefine global variables or reset controllers to default values.

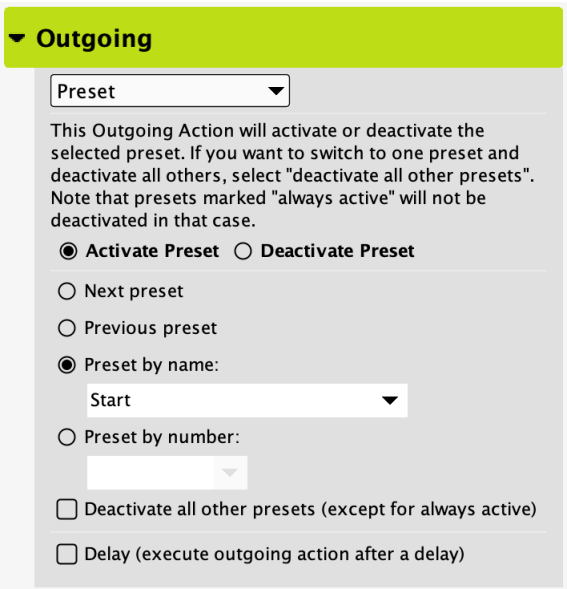


When loading or restarting a Project, a *Preset is Activated* event is triggered for all initially active presets.

Very useful is to use the *Current preset* option so that you can execute certain actions whenever the owning preset is activated, and other actions when it is deactivated.

9.7.2 Outgoing Preset Action

Outgoing Preset Actions have the capability of activating or deactivating Presets, or cycling through them all as a set.



When cycling (“next preset / previous preset”), any preset marked *Ignore for Next/Previous switching* are excluded from cycling, and can therefore be used as an always active preset.

- **Next Preset / Previous Preset**
  This outgoing action will cycle through the available presets in order, activating only one at a time. Presets must be arranged in the order of which the user desires to cycle them on and off. This is normally the preferred way of cycling through presets.

- **Preset By Name**
  This action will activate/deactivate a preset that is selected by name from a drop-down box. This is useful for most simple preset setups involving few presets. Features that involve key commands can be enabled/disabled on the fly so as not to interfere with regular keyboard operation when not needed.
- **Preset By Number**
  Presets can be activated or deactivated by number, which can be specified by a unique local or global variable (see the rules section for more information). This outgoing preset action is useful for more complex MIDI Translator projects that have many presets and change them on-the-fly depending on other Translator settings and variable states. Note that the first preset has number 0, the second preset number 1, and so on.

- **Deactivate all other presets (except for always active)**
  Presets also have the checkbox option available to 'Deactivate all other presets (except for always active)'. This can be enabled in any Preset Change outgoing action to automatically disable all other presets (except, of course, the 'always active' presets). With that function, you can easily cycle through a set of presets.

9.8 Project Action

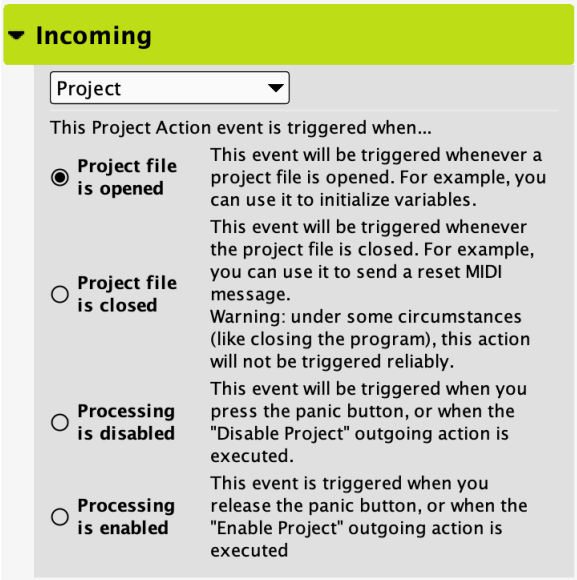
The Project Action is used to react to project wide events, and to temporarily interrupt processing on a global level.

The Project Action is fully working in translation projects running on the BomeBox.

> Note: this action was named “Disable/Enable Processing” in older versions of MIDI Translator Pro.

9.8.1 Incoming Project Action

This Incoming Action is triggered in response to internal program events.



The *Project file is opened* event occurs after the current project file is loaded. It is triggered as first action of a given project. Any actions that need to take place first before any other actions, and only once, are best defined using this incoming action. For example, it's a good idea to initialize variables to start-up values when the project file is opened.

This “Project file is opened” event transmits the following information in local variables:

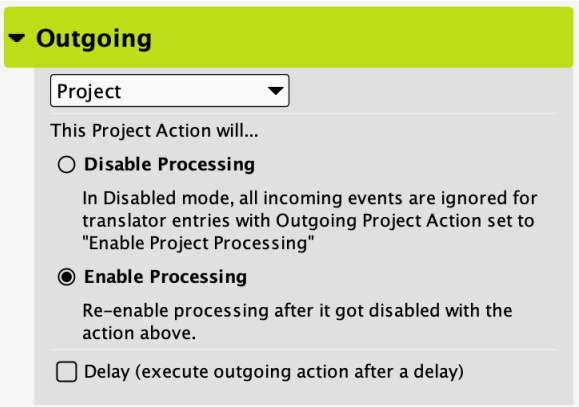
Variable `oo`: Operating System	
1	Windows
2	macOS
3	BomeBox
4	Linux
Variable `pp`: OS major version (e.g. for Windows 11.2, it is 11)	
Variable `qq`: OS minor version (e.g. for Windows 11.2, it is 2)	
Variable `tt`: Full or Demo version	
0	Full version
1	Demo version

The *Project file is closed* occurs when closing the current project file via the menu, or before loading a new project file, or when closing the program. For example, you can use it to send reset MIDI messages. Be aware that under certain situations, the *Project file is closed* event may not get triggered, for example when the computer is rebooting.

The *Processing is disabled* type is triggered when you press the panic/STOP button, or when the *Disable Processing* Outgoing Project Action is executed. You can use it to send additional STOP messages or to do other housekeeping.

Conversely, the *Processing is enabled* type is triggered when processing restarts by either pressing the STOP button again, or when the outgoing Project Action *Enable Processing* is executed.

9.8.2 Outgoing Project Action



There are two types:

1) Disable MIDI Translator Processing, and

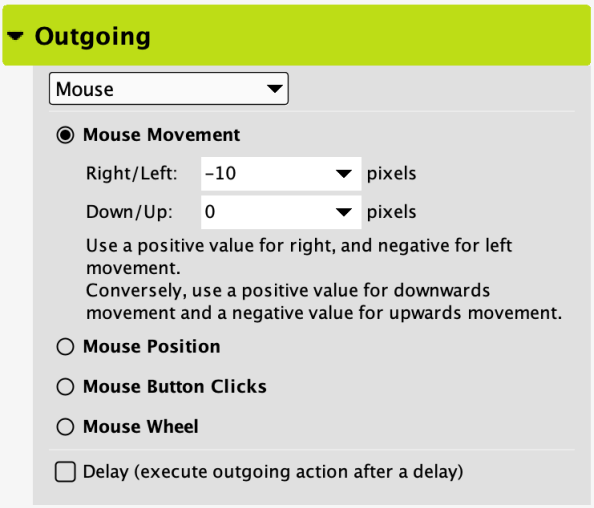
2) Enable MIDI Translator Processing.

The MIDI processing engine as a whole can be bypassed using the *Disable Processing* Outgoing Action, preventing any of the translators from being executed. Incoming events will not trigger any Outgoing Actions. There is one exception to this rule: all Translators with an Outgoing Action of type *Project* are still executed.

During *Disable Processing*, the only way to re-enable it from the project is by executing the *Enable Processing* Outgoing Action defined in a different Translator. As said above, all such Translators with Outgoing Project Action remain enabled. You can also re-enable a disabled project by pressing the Panic button in the tool bar.

9.9 Mouse Action (Outgoing)

The Mouse Outgoing Action allows different aspects of the computer mouse to be emulated: pointer movement, absolute positioning, button clicks and mouse wheel.



Each of those mouse actions includes settings for a variety of different parameters that can be manipulated to control the system mouse.

The Mouse Action does not work in translation projects running on the BomeBox.

9.9.1 Movement

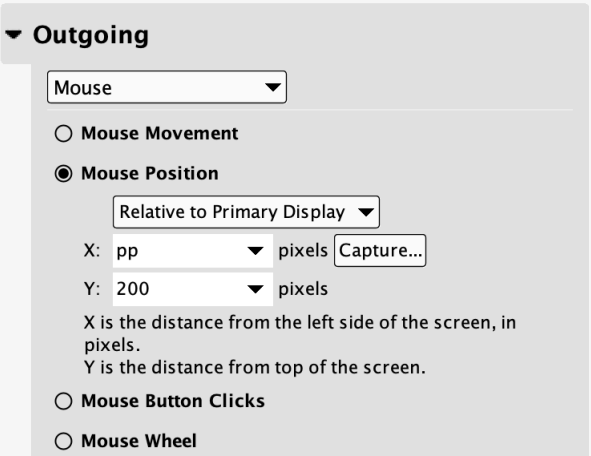
The Mouse Movement action will move the system mouse pointer. Movement is defined using two text boxes, one for Up/Down movement and the other for Left/Right movement, for specifying the respective mouse pointer movement in pixels. For Up and Left movement, use negative numbers. For Down and Right movement, use positive numbers.

The example above will move the mouse pointer by 10 pixels to the left.

You can also use variables for movement.

9.9.2 Absolute Position

The absolute positioning mouse action allows for the mouse to be moved to a predetermined point on the screen. Absolute positioning is measured in pixels, similarly to how screen resolution is set. Use your system's screen resolution as a guide for setting absolute positioning (example: if you have a 4000x1600 screen, the exact center of the screen would be absolute position 2000x800).



The X coordinate is the distance from left, and the Y coordinate is the distance from top of the screen. Therefore, specifying X=0 and Y=0 will put the mouse cursor in the *upper left* corner.

You can use the Capture button to find the absolute position by positioning the mouse cursor and then clicking with the left mouse button to capture that position.

You can also use variables for the X and Y coordinates.

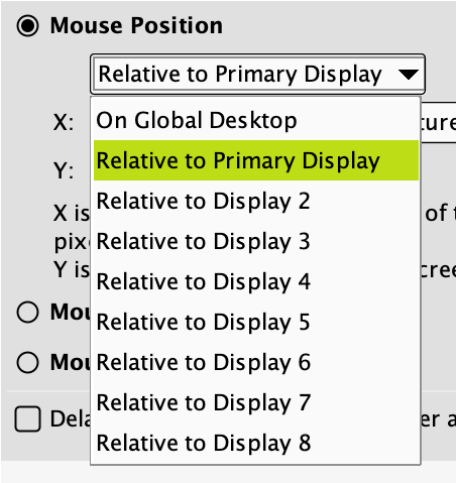
9.9.3 Multiple Display Support

Since version 1.8.4, MIDI Translator Pro allows you to target different displays individually, or treat all displays together as one virtual desktop.

In previous versions, all mouse coordinates were always relative to the primary display.

Relative to Primary Display

By default, absolute positioning of the mouse cursor is relative to the primary display. X=0 and Y=0 is the upper left corner of the primary display. If you have a second display to the right of the primary display, use X coordinates beyond the primary display to position the mouse cursor on the second display.



Similarly, if your second display is positioned above the primary display, use negative Y coordinates to position the mouse cursor on the secondary display.

On Global Desktop

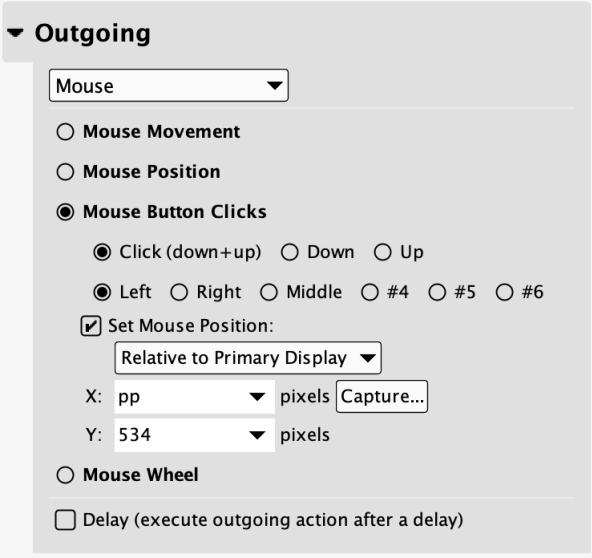
When positioning (or capturing) the mouse cursor using the Global Desktop option, all screens are considered one big monitor. So if you have one screen with size 1000x700 pixels to the left of a second screen with 500x950 pixels, the resulting desktop has the size 1500x950 pixels. Use the OS display settings to arrange the monitors as you like it.

Relative To Display 2, 3, ...

If you choose positioning relative to a specific display, the mouse is positioned on that display. Just like for the Primary display, you can reach other displays when exceeding that display's bounds.

9.9.4 Button Clicks

Button clicks can be emulated using this mouse outgoing action. By default, button click events occur at the current position of the mouse pointer. A Button Click event is comprised of a complete Mouse Down+Mouse Up event, unless otherwise selected.



You can emulate button clicks for left, right, and middle buttons as well as for auxiliary buttons 4, 5, and 6. It depends on your mouse settings what they will do. For example, the 4th button is often mapped to the “Browser Back” function, and the 5th button to the “Browser Forward” function.

Optionally, you can position the mouse before executing the click by checking *Set Mouse Position*. It works the same as *Absolute Mouse Position* (see above) and effectively executes two actions at once.

9.9.5 Wheel

Mouse wheel events can also be transmitted. Mouse wheel events can either be Forward (away from you) or Backward (towards you).

You can also specify variables for the amount wheel motion. Use a positive number for moving forward, and a negative number for moving backward. The wheel “ticks” are 1/10th of a wheel “click”, so specifying 10 will emulate moving the mouse wheel one “click” forward (away from you).

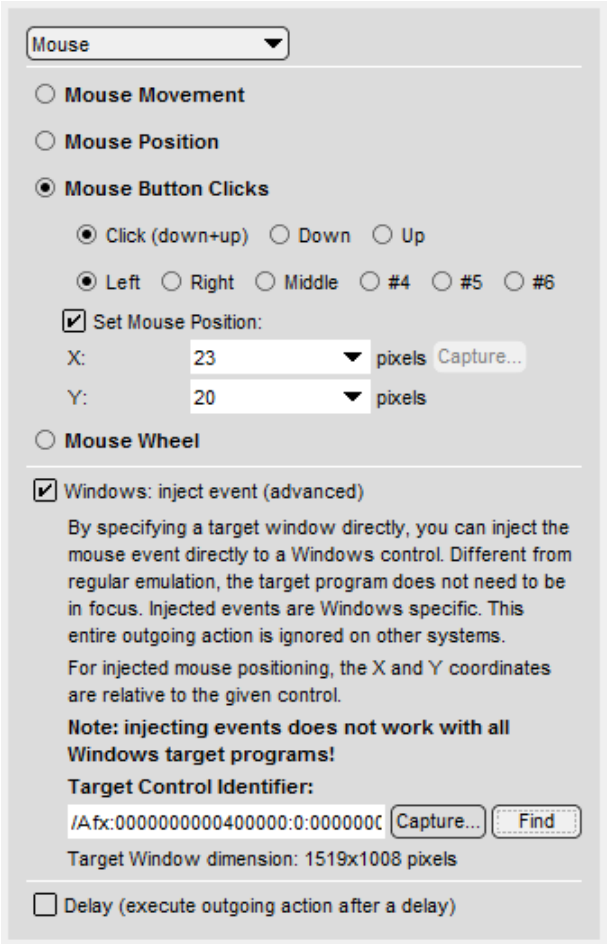
Moreover, you can emulate a second wheel, which usually maps to horizontal movement: use a positive number for “wheeling” to the right, and a negative number for the left.

9.9.6 Injected Mouse Events (Windows only)

On Windows, you can inject mouse events directly to a specific application, similar to injecting keystrokes. For programs with which this works, you can send mouse events while not moving the real mouse cursor.

> Injecting mouse events only works on Windows. It does not work with every program, however, and you will need to test with your application of choice.

Injecting mouse events can make your projects much more robust because it will still work when (accidentally) another program is activated, and you can control multiple programs with mouse events without the need to switch the active window.



For injecting mouse events, simply check that option in the Outgoing action, then click on the Capture button. Moving your mouse will now show a green rectangle around every recognized window and sub window. Click on the window or sub window you want to target with the injected mouse event. Use the Find button to see if the window definition is sufficient to find the corresponding window.

When injecting mouse events, all X/Y positioning is with regards to the targeted window! X=0 and Y=0 is therefore the upper left corner of the target window.

We recommend that you always test if the injected mouse event actually works. Because most of the time, application windows are layered, you may need to experiment a bit to find the suitable sub window for injecting mouse events.

Sometimes, it works to directly inject, say, a mouse button click event to a button sub window, effectively clicking the button.

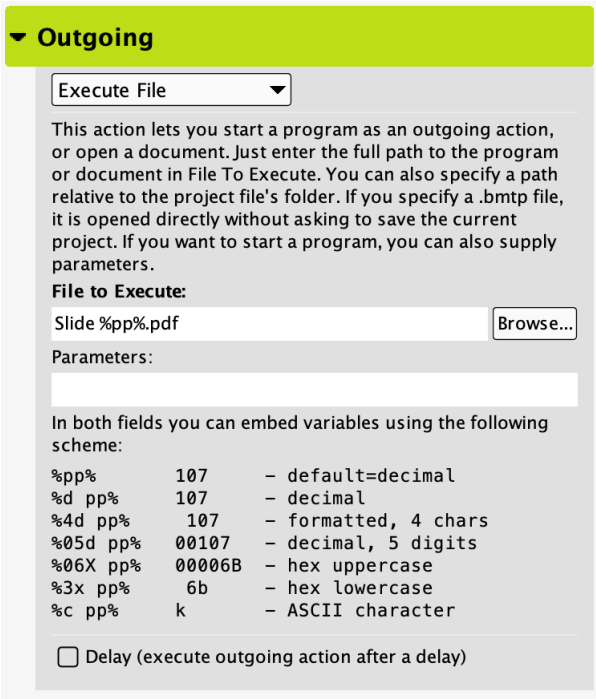
In this example, the injected mouse button press clicks the TAP button of an Ableton Live window.

Note: Injected mouse events cannot be targeted to a specific monitor.

9.10 Execute File Action (Outgoing)

The Execute File outgoing action type lets you define an executable file with parameters to run as a resultant outgoing action. Enter the executable file name with path in the File Name text box, along with any parameters you would like to pass to it in the Parameters text box. Parameters should be surrounded by quotation marks.

The Execute File Action is working in translation projects running on the BomeBox, but only for opening a project file (see below).



Start a Program

If you enter the name of a program in the File to Execute, that program will be started when this Outgoing Action is triggered. Programs will typically have the `.exe` extension on Windows and `.app` on macOS.

When starting a program like this, you can pass parameters to the program, as specified in the Parameters field. For example:

File to Execute:
`/usr/bin/wget`

Parameters:
`-o "/Users/bome/Documents/index.html" "https://www.bome.com"`

Opening Documents

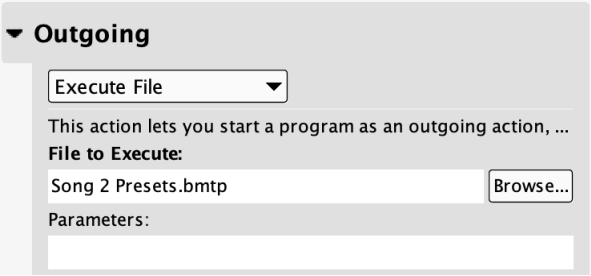
In the *File to Execute* field, you can specify a document instead of a program (see the screenshot above). In that case, the associated program will open the given document. For example, you can enter the filename of a PDF document in the Filename field. Now executing this action will start the default PDF viewer and let it open the PDF document. In this case, any given Parameters are ignored.

Relative Filename path

If the file to be executed is located in the same folder of your project file, or in a sub folder, it's recommended that you use a relative filename (e.g. `"apps/MyApp.exe"`). That way, it will be much easier to use the preset on a different computer.

Opening a MIDI Translator Project File

If you specify a .bmtp file in the Filename, it will be directly opened in the running instance of MIDI Translator, unloading the currently running project file first.



Using Variables in Filename and Parameters

You can include the value of variables in the filename and in the parameters using the %..% scheme:

Number Representation:

Normal (decimal) value of a variable: `%var%` or `%d var%`
e.g.: `%pp%` will insert the value of the variable pp.

Hexadecimal value of a variable: `%x var%` or `%X var%` (upper case)
e.g.: `%X pp%` will embed the uppercase hexadecimal value of `pp`

Letter (ASCII character): `%c var%`
e.g. `%c pp%` will embed one letter, corresponding to the ASCII value of `pp`.
If `pp` has value 65, a lower case `a` is replaced.

Number Formatting:

You can specify the minimum length of the formatted number. Just specify the length in digits before the format specifier: `%4d pp%` will embed the decimal value of pp with at least 4 characters. If the number `pp` has less than 4 digits, the embedded value is prefixed with as many spaces to make the embedded value 4 characters long. If the value of `pp` has more than 4 digits, the resulting number will be embedded with all (more than 4) digits.

e.g.: `%3d pp%`, and `pp` is 51, will result in `“ 51”`
(note the space in front of “51”)

Prefixing the length with 0 will use zeros for prefixing, if necessary.

e.g.: `%05d pp%`, and `pp` is 51, will result in `“00051”`

Number formatting works with the `d`, `x`, and `X` format specifiers.

Example:

PDF slide viewer with random access to specific pages from MIDI Translator. You can use this outgoing action in conjunction with an Incoming Action, e.g. a MIDI controller, to display arbitrary slides.

```
File:              slides/slide%02d g0%.pdf
Parameters:        (empty)
```

Now you can define which PDF file to show, based on the variable `g0`. If `g0` is 0, the PDF viewer is called like this:

```
         PDFViewer slides/slide00.pdf
```

If, later on, `g0` is 13 (by way of Rules or the Incoming Action), it is called like this:

```
         PDFViewer slides/slide13.pdf
```

So, by creating single PDF files for every page/slide, you can control exactly which slide to show with the PDF viewer (and jump to particular pages, etc.).

9.11 Serial Port Action

With the Serial Port Incoming and Outgoing actions, you can trigger your translators from arbitrary data received on a serial port, and send any data you like on a serial port.

So you can easily implement a Serial-to-MIDI converter, and vice versa, with just a few Translators, or convert the serial data flow on the fly, or control devices which only have a serial port.

The Serial Port Action is fully working in translation projects running on the BomeBox.

9.11.1 Data Representation and Format

The serial port actions let you enter the Incoming or Outgoing serial port data string in three different ways.

ASCII Text

In ASCII mode, you can enter arbitrary text into the Serial Port text field, and it will be interpreted as a series of ASCII characters. Many terminal type serial port devices and modems (initially) work in ASCII mode.

You should only use characters from the original ASCII type set. Special characters and control codes can be entered by escaping them with backslash: for ENTER, use \r\n or just \n. For entering a single backslash, use \\. For entering an arbitrary byte, prepend the hexadecimal number with \x. For example, to send a byte 175 ('AF' hexadecimal), use `\xAF`. The hexadecimal number must have 2 digits. For entering longer binary data, specify them byte for byte, e.g.:

```
\xFF\xF0Embedded Text\x00\x0A\x08\xF7
```

In ASCII mode, you cannot embed variables.

Data (numbers)

In this mode, you specify the serial port data as a series of bytes, separated by a space. One byte is in the range 0 to 255. You can also specify hexadecimal numbers by prefixing them with 0x. Local and global variables can be embedded directly.

Example:

```
255 0x7F 10 20 pp g0 10
```

In the example, the first byte is specified as a decimal number 255, the second number in hexadecimal form, which equals 127 decimal. Then two more decimal numbers, followed by the two variables pp and g0.

For incoming serial port actions, embedded variables are set to the received byte at that position. For outgoing serial port actions, variables in the outgoing string are replaced by their values before being sent `ot` the serial port.

Data (hexdump)

This way to specify the serial port data string is similar to Data (numbers), but the data is entered as a series of hexadecimal bytes without a prefix.

You can embed variables by enclosing them in % signs.

Example:

```
Data (hexdump): F0 60 %pp% F7
```

This is equivalent to specifying the string like this in Data (numbers) mode:

```
Data (numbers): 240 96 pp 247
```

9.11.2 Selecting a Serial Port and Alias

In every Serial Port action, you need to specify the serial port to receive data from, or to send it to, respectively. However, most often serial ports have non-descriptive names like `COM12` or `/dev/cu.Bluetooth-Incoming-Port`. Therefore, MIDI Translator enforces the use of Serial Port Aliases. They are similar to MIDI Port Aliases, and basically just define an own name for the port. We recommend using the name of the connected device for the alias name. This makes it easier to use your project on a different computer where the actual serial port might be “COM10” and not “COM12”.

When you open a project file with unknown serial port aliases, a pop-up dialog will prompt you to select which serial port a given port alias is assigned to.

Consequently, to use a serial port in an incoming or outgoing action, first use the *Create* button to create an alias, then check that alias in all the serial port actions you need.

Under some circumstances, it might even make sense to create multiple aliases for the same port.

To define the physical port to an alias, click the *Edit* button. In order to remove an alias, use the *Delete* button. If the alias is used in other translators, it will likely be recreated. Also, if you load a project file with the deleted alias, it will be recreated.

The alias assignments are stored on a local computer. When using a project with serial ports on a different computer, you will first need to define the serial port aliases.

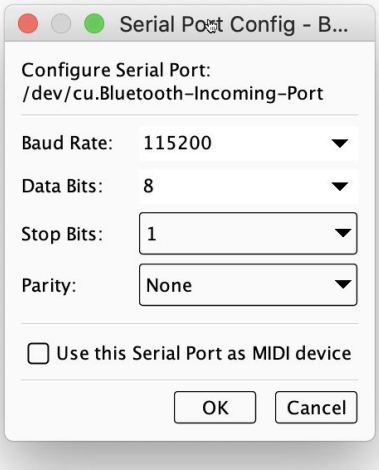
9.11.3 Configuring the Serial Port

By pressing the Configure button, you can specify the port's connection settings.

> Serial port configuration can be done on both the actual serial ports, and on serial port aliases.
> We recommend that you only configure port *aliases*.

The port configuration is stored in the project file so you can use different port configurations in different projects.

Note: prior to version 1.8.2, you could only set the configuration on serial ports and not on serial port aliases.



Baud Rate

Use the drop-down list to use one of the usual baud rates, or enter a custom baud rate directly. Note that some operating systems and/or serial port devices may not support all baud rates. The most common rate for terminals and the like is 115200. Physical MIDI baud rate is 31250.

Data Bits / Stop Bits / Parity

These definitions specify the physical protocol. They depend on the receiving serial port device. Most common is 8N1, i.e. 8 data bits, parity None, and 1 stop bit.

Use as MIDI Device

By checking this option, MIDI in and out ports appear inside MIDI Translator. See the next chapter for details.

9.11.4 Using a Serial Port as a MIDI Device

There are certain devices which send MIDI data over a standard serial port. For those scenarios, you can check the option “Use as MIDI Device” in the serial port configuration. This will create a MIDI device internal to MIDI Translator, as a direct pass-through: any MIDI messages sent to that MIDI OUT port will actually send it to the associated serial port. And any data received on that serial port will also be received on the corresponding MIDI IN port.

This function also allows you to use serial ports in the MIDI Router.

Mixed use is possible, too: even if exposed as a MIDI device, the serial port can still be used in serial port action.

We recommend that you only use this function on serial port aliases. The MIDI port will be named like the alias.

The “Use as MIDI Device” setting on a serial port (alias) is stored in your project file. (Prior to version 1.8.2, it was stored on the computer instead).

9.11.5 Capturing Serial Port Data

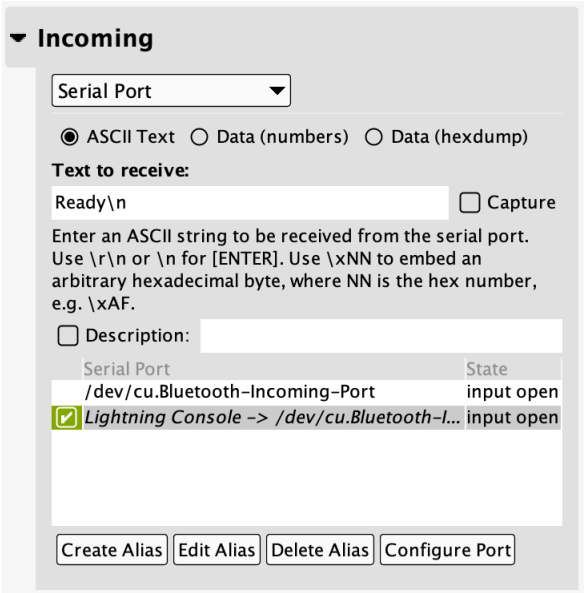
By activating the Capture checkbox next to the serial data text field, all data received from the opened serial ports will be appended to the text field. This allows for a convenient way to define the data string you need.

> The Capture function always displays received serial port data, not the data which is sent out by an outgoing action.

9.11.6 Incoming Serial Port

In the incoming serial port action, you define the serial port string that acts as a trigger for the translator. The string you enter is matched anywhere in the data string being received from the serial port, so you can match for partial lines, single characters, and multiple lines all the same.

See above for the different ways to enter the serial data string.



For the Data modes, you can embed variables. Upon a matching serial port string, the variables will be set to the received data byte at the given position. As long as the incoming action does not match, the variables are not touched.

Example:

Incoming Action data string in Data/hexdump format:

```
6E 20 %pp% 70 %ga% 5F
```

Now let this be a series of data bytes received on the serial port:

```
... 20 6E 6E 20 3A 70 51 5F 80 1A ...
```

As soon as the byte 5F is received (last byte of the incoming action), the incoming action matches. The variable `pp` is set to hexadecimal 3A (decimal 58), and the variable `ga` is set to the value hexadecimal 51 (decimal 81).

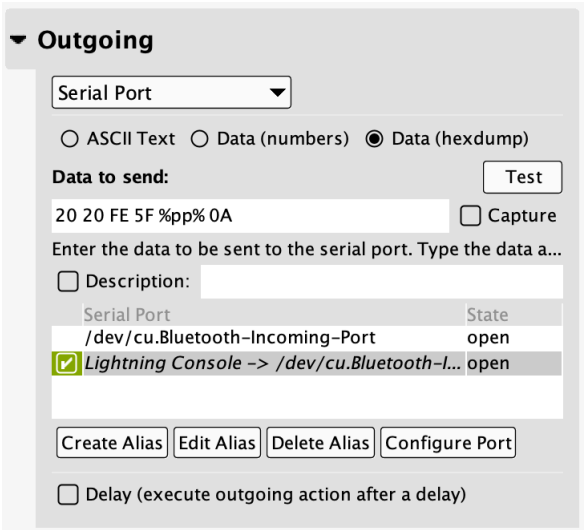
Now, in the rules and outgoing action, you can use and modify these variables, or, for global variables, use them in different translators.

9.11.7 Outgoing Serial Port

This action is used to send a data string to a serial port.

See above for a description of the different ways to specify the data string.

For the Data modes, you can embed variables. Before the string is sent out, the values of the variables are inserted instead of the variables. The variables will not be changed.



Example:

Outgoing Action data string in Data/number format:

```
10 20 30 pp 67 0xFF h1 0
```

Now let variable pp be 56 and h1 be 126. Then this is the string to be sent out:

```
decimal:          10 20 30 56 67 255 126 0
hexadecimal:      0A 14 1E 38 43 FF 7E 00
```

9.12 AppleScript Action (Outgoing)

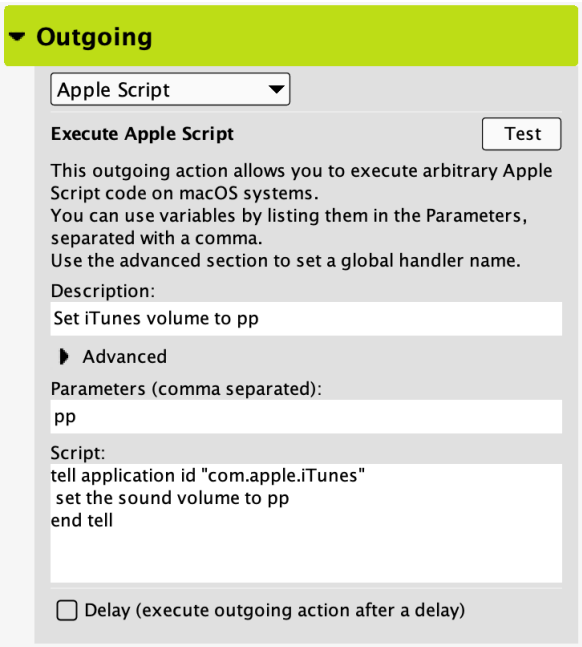
The AppleScript action allows you to execute arbitrary AppleScript fragments when running on macOS.

The AppleScript Action does not work in translation projects running on the BomeBox.

9.12.1 AppleScript Outgoing Action

You can invoke arbitrary AppleScript commands as an Outgoing Action.

> Although you can edit the AppleScript action on Windows, it will only execute on macOS.



The AppleScript code that you enter in the Script text area will be executed when this Outgoing Action is triggered.

For documentation, you can enter a **Description** text. It does not change the functionality.

If you want to access MIDI Translator variables in your script, use the **Parameters** field (see section Passing Variables directly to a script below).

In the Advanced section, you can optionally specify a (unique) **Handler Name**, if you want to call this AppleScript code from other AppleScript sections. See section Specifying a Unique Handler Name below.

You can also define AppleScript handlers which you can call from anywhere: define them in the **Global AppleScript** section of the Project Properties. These handlers are available in all AppleScript actions (see Project Global AppleScript Section below).

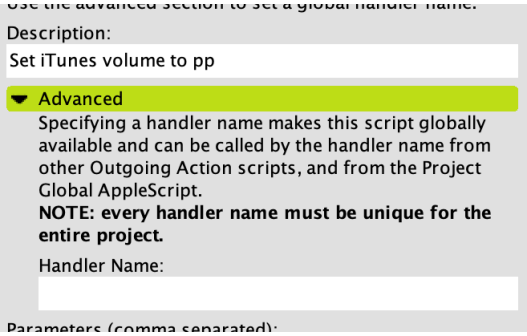
Passing Variables directly to a script

Any variables that you want to use in the script can be passed as parameters. In the Parameters text field, enter all local and global variables that you want to read in the script. For example, by specifying “pp, g0” as Parameters, you will be able to use pp and g0 directly in the AppleScript. See the screenshot above for an example.

> The Script will use copies of the variables so any changes to the variables in AppleScript code will not be reflected back to the MIDI Translator

engine. This is only possible via the setVariable() handler (see Modifying MT Variables below).

Specifying a Unique Handler Name



Under the Advanced drop-down, you can optionally specify a handler name for the Outgoing AppleScript action.

If you do that make sure to use a unique name: you must not use the same handler name anywhere else in an Outgoing Action or in the Project Global AppleScript section.

As a rule of thumb, never set the **handler name** of an Outgoing Action unless you want to call it from another AppleScript section.

Call Outgoing AppleScript from other Outgoing Actions

The AppleScript outgoing action is internally implemented as a handler. For that you must specify a unique handler name in the Outgoing Action. Now that handler is also available from other AppleScript actions and from the global AppleScript section.

Modifying MT Variables

From inside AppleScript code in the Outgoing Action, you can access global variables directly:

```
setVariable(<Name>, <value>)
getVariable(<Name>)
```

For this to work, you must have installed MIDI Translator in the *Applications* folder.

Example: increase iTunes volume:

```
-- increase volume, stored in g0
set vol to getVariable("g0")
set vol to vol + 10
setVariable("g0", vol)
tell application id "com.apple.iTunes"
  set the sound volume to vol
end tell
display notification "The volume is now: " & (g0 as text)
```

Project Global AppleScript Section

In the Project Properties, there is a text area where you can enter free form AppleScript handlers. These handlers can be called from your outgoing action so that you have a space for AppleScript code that you need to run from multiple outgoing actions.

Example: global handler for iTunes volume

The following text, when entered in the global AppleScript section, defines a handler that lets you set the iTunes volume.

```
on setITunesVolume(vol)
  tell application id "com.apple.iTunes"
    set the sound volume to vol
  end tell
end setITunesVolume
```

Now from any Outgoing AppleScript action, you can call the handler `setITunesVolume(val)`.

> Do not enter code outside of a handler in the global section. It will be executed at arbitrary times (or not at all).

Referencing MIDI Translator in an AppleScript

The special identifier __APPLICATION_NAME__ is replaced with the application name, i.e. “Bome MIDI Translator Pro” (or to whatever you have renamed the program bundle name in the Finder). If you need to self-reference the MIDI Translator application, use that identifier rather than hard coding the name.

9.12.2 Control MT using External AppleScript

You can control a running instance of MIDI Translator from 3rd party AppleScripts.

Currently, there are 2 commands available:

```
        set variable <varname> to <value>
        get variable <varname>
```

You can only manipulate global variables.

For external AppleScript support to work, you must have MIDI Translator installed in the *Applications* folder.

For example, the following script sets the global variable g0 to 123, and then reads it back and sets an AppleScript variable `asVar` to the result:

```
tell application "__APPLICATION_NAME__"
        set variable "g0" to 123
        set asVar to get variable "g0"
end tell
log asVar
```

Also note the use of the __APPLICATION_NAME__ constant instead of “Bome MIDI Translator Pro”.

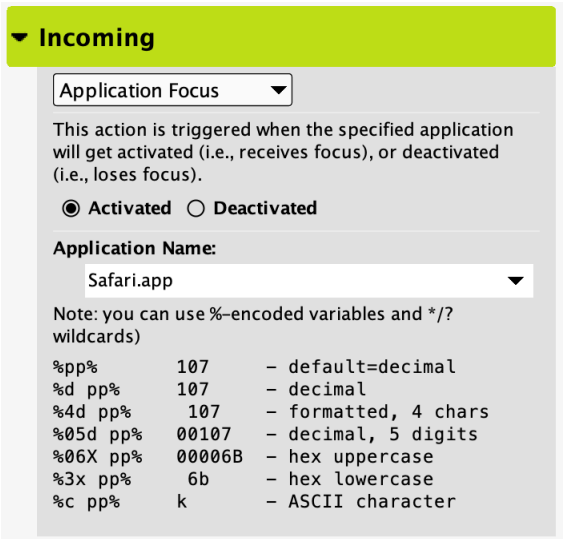
9.13 Application Focus Action

This action type manages the focus of other applications. An application has focus when it is front-most. The currently focused application is the application that receives key strokes.

The Application Focus Action does not work in translation projects running on the *BomeBox*.

The Application Focus action was introduced in MIDI Translator Pro version 1.9.

9.13.1 Incoming Application Focus



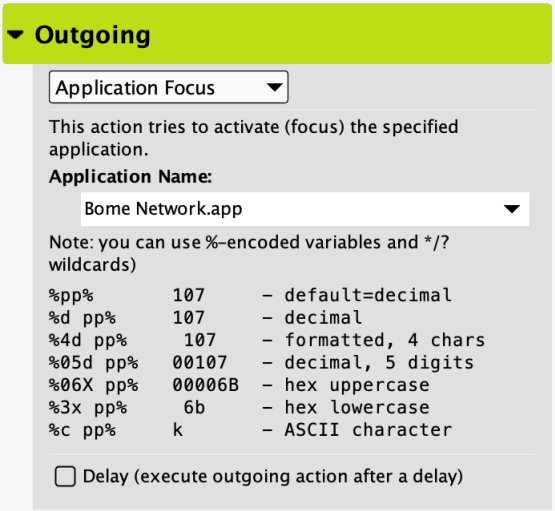
You can use this Incoming Action in order to detect when the user switches to a different application. For example, you can enable a certain preset when Ableton Live is active, and switch to a different preset when Safari is activated.

The screenshot above shows an Incoming Application Focus action which listens for Safari to become the front-most, focused application (*Activated*).

Conversely, you can also define the incoming Application Focus action to react when an application looses focus (*Deactivated*).

For defining the *Application Name*, use the drop-down list to select an application name from the list. This is generally the filename of the .exe / .app without path of the application. You can use wildcards and %-encoded variables in the application name, too. For example, if you specify `S*` as Application Name, this Incoming Action always triggers when an application gets focus, where the application name starts with an S.

9.13.2 Outgoing Application Focus

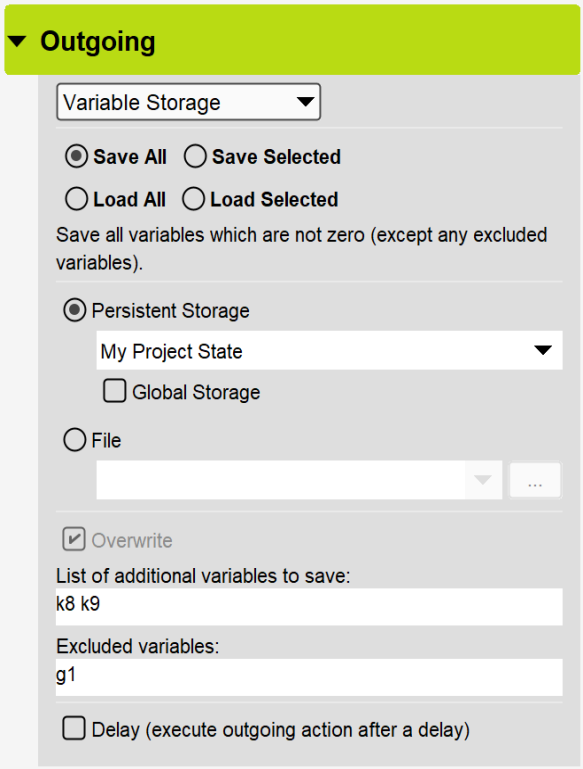


The Outgoing Application Focus action is used to forcefully put a given application to front. In the example above, the Bome Network application is put to front when that Outgoing Action is invoked.

Just like in the Incoming Application Focus Action, you can use wildcards and %-encoded variables in the Application Name. If multiple applications exist which match the Application Name, MIDI Translator Pro selects one at random.

Note that the operating system may not allow forcing the application focus under certain circumstances, so be sure to test the functionality.

9.14 Variable Storage Action



This Outgoing Action allows you to save the value of one or more global variables, and to retrieve their value again later. The storage is persistent, so you will be able to restore variables saved before restarting MIDI Translator Pro, or restarting the computer.

Variable Storage requires:

- Bome MIDI Translator Pro: version 1.9.2 or later
- BomeBox: firmware version 1.6 or later

For example, you can save the state of all variables, so that the next time you open the project, you can load the values of those variables and continue your project exactly where you were when saving the variables.

Or you can save or update just a few individual variables, e.g. for options that define behavior of your project file and should be persistent.

The Variable Storage Action is invoked just like any other Outgoing Action, so you can Save or Load variables in response to starting up the project, when enabling or disabling a preset, in response to MIDI messages or keystrokes, or as part of a *Perform* translator.

Global Variables

You can only save or load Global Variables from Variable Storage.

Persistent Storage

The standard way to save and load variables is the *Persistent Storage*. Each storage "slot" is identified by a name that you choose. Bome MIDI Translator Pro takes care of where the variables are actually stored.

Persistent Storage works well in Bome MIDI Translator Pro on all platforms, including the *BomeBox*.

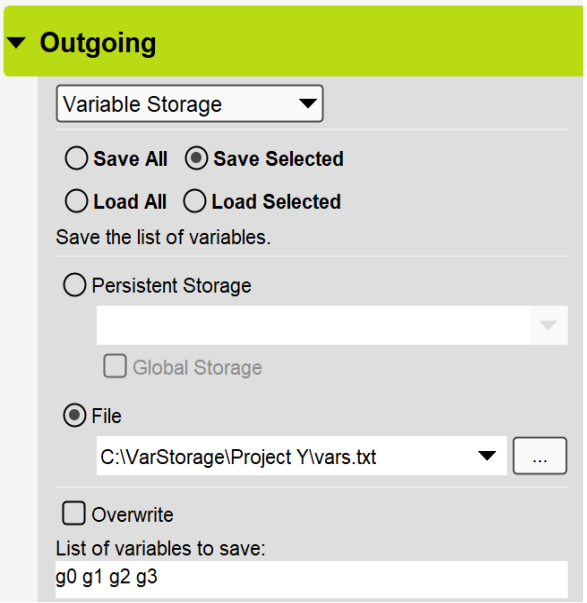
Option: Global Storage

If the *Global Storage* option is not selected, then the persistent storage is private to this project file. For example, if you specify a persistent storage name "My Project State", you can save and load variables from this storage slot. However, a different project file can

also use a persistent storage “My Project State”, but it will use its own (private) slot. Non-global, private variable storage uses the project file name to identify the storage slot, so two project files with the same project file name will also use the same private storage.

Conversely, all storage slots marked *Global Storage* share their named slots with all MIDI Translator Pro projects. So you can save variables to storage with name “A Global State” from one project, and load this same storage from a different project with a different name. On the other hand, make sure to not use the same storage name in different projects unless you *want* to share it across the different projects.

File Storage



Alternatively, you can save and load variables to/from a file on your hard disk. This is a more advanced option. The internal Persistent Storage is the default option.

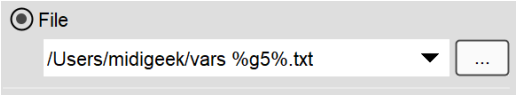
You can specify any name you like, but we recommend a filename extension `.txt`.

If you use an **absolute path** in the filename, the file will be saved or loaded to/from the given path (at the right side you can see an example of an absolute path on Windows). If folders in the given path do not exist, they are created when saving the file.

Note: absolute paths will be converted to relative paths when running a project file on the BomeBox.

If you use a **relative path** (i.e., a path that does not start with /, \, or a drive), the file will be saved/loaded relative to the location of the project file. If you don’t provide a folder at all (just a filename), it will be saved/loaded along side the project file.

Using Variables in Storage Name



For both Persistent Storage names and filenames, you can embed variables in the name. The variables are evaluated at the time of executing the Outgoing Action. For more information, see *Using Variables in Filename and Parameters* on page 75. In the example on the right side, the file will be saved under different names depending on the value of the variable `g5`. For example, if g5 has the value 20, the resulting filename is:

```
/Users/midigeek/vars 20.txt
```

Timing

Loading and saving variables can take some time (in milliseconds), because the persistent storage is saved on the file system. The Variable Storage action is usually executed synchronously, so that any following Translators for the same incoming event will be delayed. Usually, saving or loading only takes a few milliseconds, but for real-time MIDI performance, that can already be a problem.

If you want to avoid this performance hit, you can set a very small delay for the Outgoing Action (see Delaying Outgoing Actions) to force the Variable Storage action to be executed asynchronously.

File Format

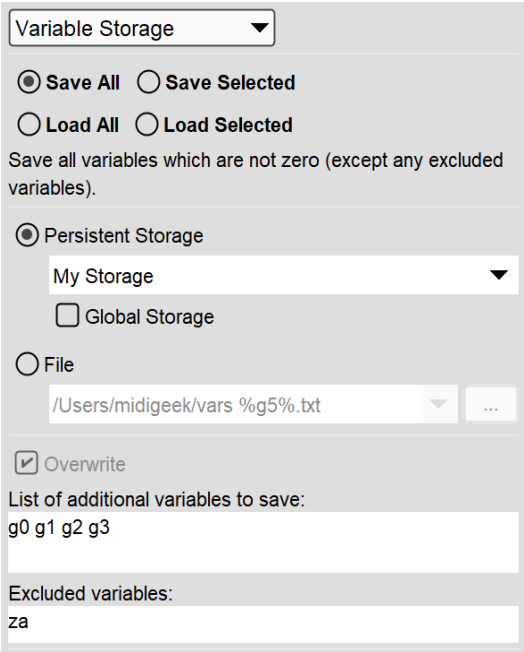
When saving to a *file*, the variables are written to a text file. The file format is ASCII text, consisting of one variable definition per line, e.g.:

```
g0=10
g1=12
g2=-652
g3=1234
```

9.14.1 Save All Variables

When selecting *Save All* in the Outgoing Action, triggering the action will save all variables which are currently set with a non-zero value to storage or file. This is a convenient way to store all variables that you have used, considering that at start of a project, all variables are initialized with 0.

Save All: save all non-zero variables.



If you want to make sure that certain variables are always saved in the persistent storage (even if their value is 0), then you can specify additional variables in the *Additional variables to save* field. In the example at right, the variables `h1` and `k2` are always saved along with all non-zero variables. This can be useful if you are using the Load action without clearing all variables first.

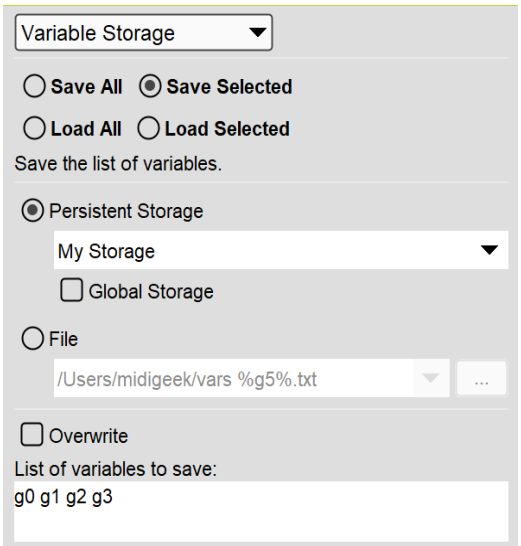
Conversely, if there are certain variables that you don't want to save even though their value is non-zero, specify these variables in the *Excluded Variables* field.

Note that when saving all variables, the storage or file is always overwritten. Any variables stored under the same storage name / filename are overwritten by the new saved set of variables.

9.14.2 Save Selected Variables

When selecting *Save Selected*, only the specified variables will be saved to the given storage or file.

The *List of variables to save* field allows you to enter all the variable names that you want to save. You can use multiple lines, and separate them by space or comma. In the example at right, the variable storage will be saved containing the values of exactly four variables (`g0`, `g1`, `g2`, and `g3`). These four variables will be saved even if they have not been used before, in which case they are saved with value 0.



Overwrite

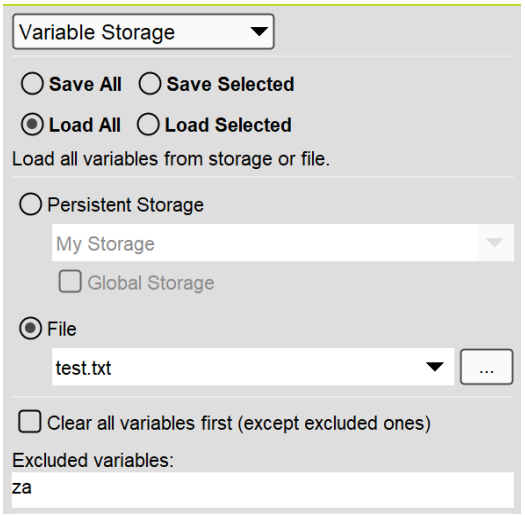
Usually, for Save Selected, the variables are saved to the storage / file, keeping other variable definitions in the storage/file untouched. So you can use this function to selectively update individual variables in the storage or file.

If you select *Overwrite*, then the entire storage or file will be newly written, erasing any existing variables from the given storage or file.

9.14.3 Load All Variables

Selecting *Load All* in the Outgoing Action means that when the action is invoked, all variables are loaded from the named storage or file. All variables means all variables which present in the given storage or file.

You can specify a list of variables in the *Excluded Variables* field. These variables will not be touched, even if they exist in the given variable storage or file. In the example at right, the variable *za* will not be loaded.



Clear All Variables First

Check this option if you want to set the variables to 0 before this action loads the variables from the given storage set or file. This can be useful in conjunction when using the *Save All* operation.

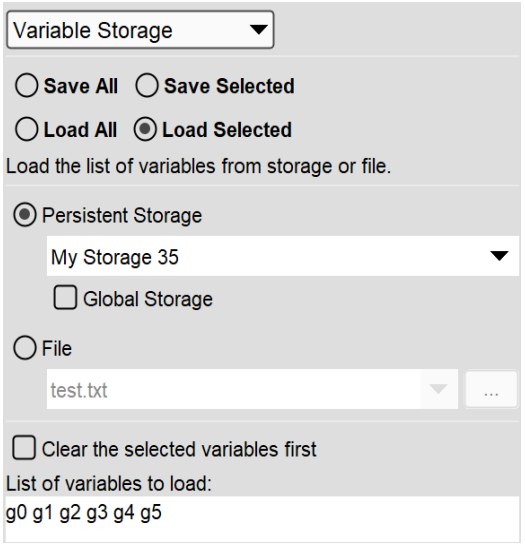
When loading all variables, checking this option will clear all global variables *only if the specified storage or file already exists* when loading. If the storage or file does not exist, variables are not cleared.

Additionally, all variables listed in the *Excluded Variables* field will not be cleared.

If *Clear all variables first* is not checked, all global variables keep their values, except those that are loaded from the given storage or file.

9.14.4 Load Selected Variables

If you have selected *Load Selected* , you can pick the variables which you want to load from the storage or file. In the example at right, only the variables g0...g5 will be loaded from the given storage. All other variables are not touched. Also, if a variable is specified here, but is not present in the storage or file, it will also not change its value. So, a variable will only be loaded here if it is
a) listed in the *List of variables to load* field, and
b) if it exists in the given storage or file.



Clear the Selected Variables First

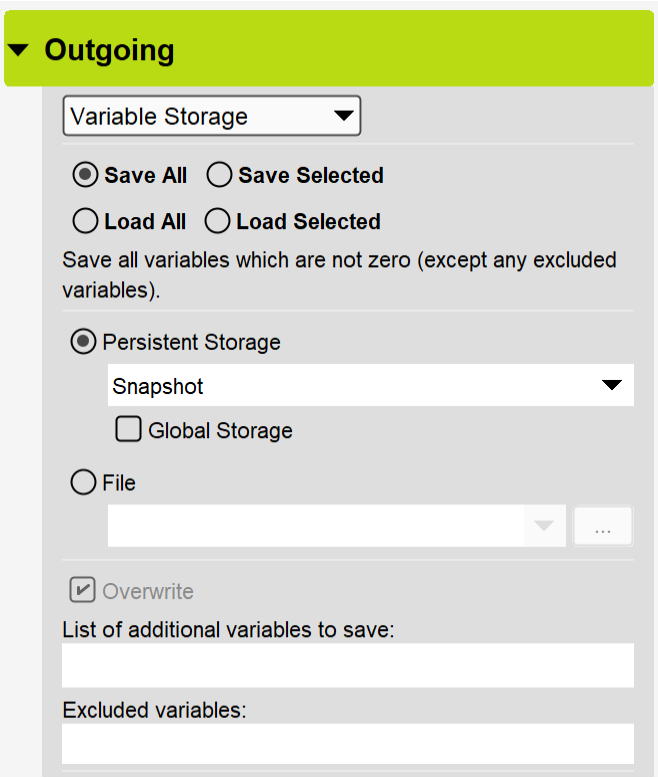
Check this option if you want to set the *listed* variables to 0 before this action loads variables from the given storage or file.

When loading the specific variables, checking this option will clear these variables *only if the specified storage or file already exists* when loading. If the storage or file does not exist, the specified variables are not cleared.

If *Clear the selected variables first* option is *not* checked, the variables will only change their value if they exist in the given storage or file.

9.14.5 Example: Saving and Loading All Variables

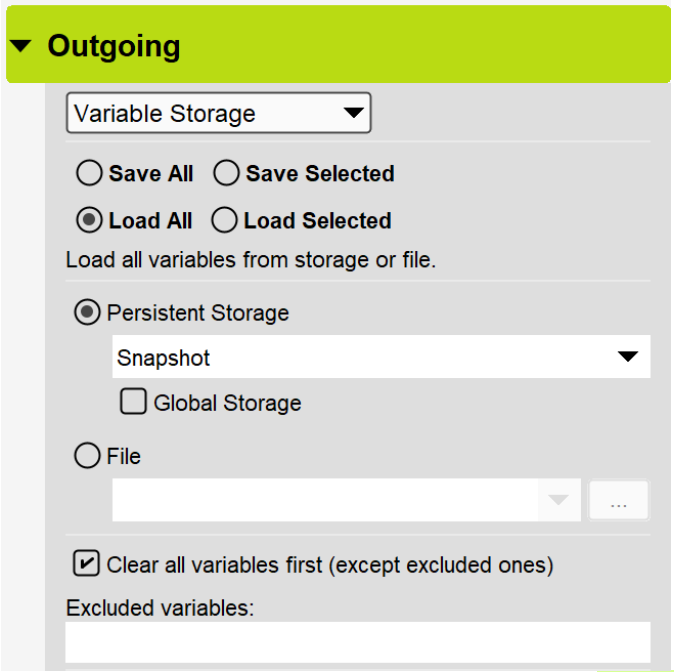
If you want to store a snapshot of all variables, you can define a Translator Entry that saves the entire set of variables. For example, you can define a hotkey (like Shift-Ctrl-F12) as incoming action. Then define the Outgoing Action like this:



This outgoing action will save the entire variable state: all variables with a value other than 0 will be saved to the given persistent storage, named *Snapshot*.

Now we want this snapshot to be loaded whenever MIDI Translator Pro starts this project file. For that, we define an Incoming Action *On Project Start*, and as Outgoing Action the one below:

This action will clear all variables first (if the given storage exists), then load the most recent snapshot from persistent storage. All variables from the persistent storage will be initialized with the saved values.



You can define **default values** for variables before invoking the *Load* action. These values will be retained if the Variable Storage has never been saved before. That is because *Clear all variables first* will only clear the variables if the storage already exists.

9.14.6 Example: Keep a set of persistent Variables

Let's assume that a project needs to keep *g0* and *g1* persistent. The example project uses an *Init* preset to

- Initialize variables *g0* and *g1* with default values
- Load Selected variables *g0* and *g1* from Variable Storage
  - if the Variable Storage already exists, this will overwrite the default values
- Define a Perform Action `Variables Changed`. The project should invoke this Perform when one of the persistent variables was changed.
- The Perform Action starts a 5 second Timer named `Save Variables Async`.
- When the Timer elapses, it saves the selected persistent variables.

Using a timer has the advantage that if multiple variables change in rapid succession, the Save action is only executed once. If the Perform Action `Variables Changed` is invoked while the timer is already running, the timer will be retriggered so that it always waits 5 seconds after the last variable change.

Note that the Perform Action `Variables Changed` is not called automatically. You will have to add a Translator to the project that invokes the Perform when it has changed one of the persistent variables.

Of course, this scheme can be expanded to save other and/or more variables.

```
[x] Preset 0: Init
```

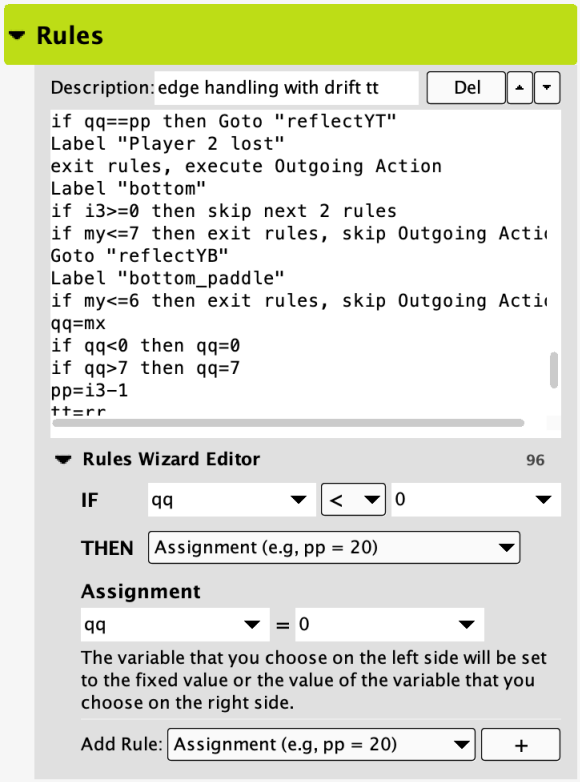
```
[x] Translator 0.0: Initialize Variables with default values
Incoming: on activation of this preset
Rules:
  g0=123
  g1=200
Outgoing: (none)

[x] Translator 0.1: Load Persistent Variables
Incoming: on activation of this preset
Outgoing: Load Selected from storage 'InitExample storage slot'
  Variables: g0 g1

[x] Translator 0.2: When Variables have changed
Incoming: On Perform "Variables Changed" with 0 parameters
Outgoing: One-shot timer "Save Variables Async": 5000 ms delay

[x] Translator 0.3: Save Variables
Incoming: On timer "Save Variables Async"
Outgoing: Save Selected to storage 'InitExample storage slot'
  Variables: g0 g1
```

10 Rules and Variables



10.1 Overview

Translators are comprised of three main sections: Incoming Actions, Outgoing Actions, and the *Rules*. This part of the documentation covers Rules, how to use and how they relate to Variables, and what they can be used for.

Rules are basically simplified programming steps that take data from the Incoming Actions, or global variables, and can affect what happens with the Translator's Outgoing Action. Rules use variables to pass data back and forth between the Incoming Action and the Outgoing Action of a Translator.

Rules are normally processed from top to bottom: the rule on the first line is processed first, followed by the second and so on. Rules can also use `Label` and `Goto` to direct programming flow. Existing rules can be moved up and down in the rules box by first selecting the rule, then clicking the 'Up' or 'Down' buttons next to the Rules list.

Variables can either be defined in an incoming action, or through the Rules section of a translator. Incoming actions defined with a variable as part of the action will pass the variable on to the rules section to be processed and potentially used as a global variable or passed on to the outgoing action.

10.2 Rule Types

There are eight types of rules in Bome MIDI Translator. Three of these rules (Assignment, Expression and Conditional) deal directly with variables, changing their values and operating off of conditionals determined by existing values. Two of the rules (Goto and Label) are used for directing the flow of the rules programming, allowing you to make 'Functions' for complex rule sets. The two exit rules (Exit Rules and Execute, Exit Rules and Ignore) are especially useful for conditionals, only enabling the outgoing action when a specific condition is met. The last rule is a comment, allowing you to document your rules in free form.

The rule types are described in detail below.

10.2.1 Assignment

Examples:

```
pp = 20
ga = qq
```

This rule type allows a straight assignment of a variable's value to a specific number or another variable's value. The variable you wish to assign is chosen on the left side of the equation from a drop-down box, while the source value or variable is selected or entered on the right side. Assignment rules are useful for assigning an input local variable to a global variable. They can also be used for assigning a specific value to an outgoing action depending on a conditional.

10.2.2 Expression

Examples:

```
pp = 30 + qq
h0 = 128 / ga
```

Expression rules use basic arithmetic (plus: `+`, minus: `-`, multiply: `*`, divide: `/`, modulo: `%`) or binary operators (AND: `&`, OR: `|`, XOR: `^`, shift right: `>>`, shift left: `<<`) to enter a value into a variable. A variable is selected from a drop-down box on the left side of the equation, while the two variables and/or numbers and operator are selected on the right side of the equation. Expression rules are useful for processing basic operators on incoming values to, for example, increase them or decrease them parametrically.

Expression Operators:

`+`	plus	Add the left operand to the right operand. Example:
		`pp=10`
		`g0=pp+110`
		→ `g0` has the value 120.
		Note: variables use a signed 32-bit range, i.e. they wrap at 2147483647 and -2147483648.

`-`	minus	Subtract the right operand from the left operand. Example:
		`pp=99`
		`g0=pp-110`
		→ `g0` has the value -11.
		Note: variables use a signed 32-bit range, i.e. they wrap at 2147483647 and -2147483648.

`*`	multiply	Multiply the left operand with the right operand. Example: `pp=2 * 22` → `pp` has the value 44. Note: variables use a signed 32-bit range, i.e. multiplying where the result exceeds 2147483647 will cause a truncated result.

`/`	divide	Divide the left operand by the right operand and truncate the decimals. Example:
		`pp=10`
		`qq=127 / pp`
		→ `qq` has the value 12
		Note: prevent a division by 0! (undefined)

`%`	modulo	Calculate the remainder of the division of the left operand by the right operand. Example:
		`pp=10`
		`qq=104 % pp`
		→ `qq` has the value 4
		Note: prevent modulo 0! (undefined)

`&`	bit-wise AND	Calculate the combination of the left operand's bits AND the right operand's bits. A given bit is set in the result when it is set in both the left and right operand. Example: `tt=0x98 & 15` → `tt` has the value 8 Note: variables and calculations use 32-bit signed integer numbers.
`|`	bit-wise OR	Calculate the combination of the left operand's bits OR the right operand's bits. A given bit is set in the result when it is set in the left operand, or in the right operand, or in both. Example: `h0=0x90 | 3` → `h0` has the value 0x93 Note: variables and calculations use 32-bit signed integer numbers.
`^`	bit-wise XOR	Calculate the XOR combination of the left operand's bits with the right operand's bits. A given bit is set in the result when it is either set in the left operand or in the right operand (but not both). Example: `h0=0x90 ^ 13` → `h0` has the value 157 (0x9D) Note: variables and calculations use 32-bit signed integer numbers.

`>>`	bit-wise shift right	Shift the left operand's bits to the right by the number of bits given in the right operand. Left bits are filled with 0 bits. Example: `ga=0x90 >> 4` → `ga` has the value 9 Note: variables and calculations use 32-bit signed integer numbers. Note: shifting right by 1 is equivalent to dividing by 2, shifting right by 2 divides by 4, then by 8, etc.
`<<`	bit-wise shift left	Shift the left operand's bits to the left by the number of bits given in the right operand. Right bits are filled with 0 bits. Example: `ga=0xB << 4` → `ga` has the value 0xB0 Note: variables and calculations use 32-bit signed integer numbers. Note: shifting left by 1 is equivalent to multiplying with 2, shifting left by 2 multiplies with 4, then by 8, etc.

10.2.3 Goto

Redirects the processing of the rules to a `Label` point. A Label name may be typed in directly, or an existing label Goto destination may be picked from the drop-down box.

Take extra care that you do not create an infinite loop!

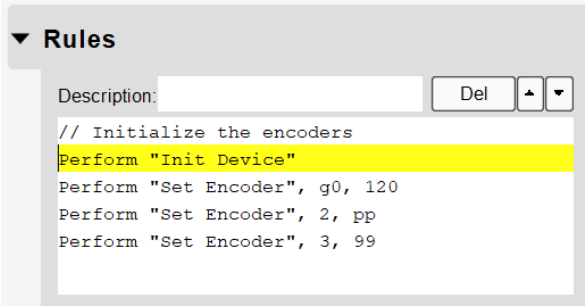
10.2.4 Label

This is the destination point in the rules processing that you would like a `Goto` point to redirect to. Labels are useful for defining functions in your rules sets.

```
// Execute a loop 10 times
pp=10
Label "Loop 1"
 xx=xx^qq
 qq=qq+1
 pp=pp-1
if pp>0 then Goto "Loop 1"
```

10.2.5 Perform

This Rule will call the given Perform immediately. Specify the Perform name in quotes. Optionally, you can pass comma-separated parameters to the Incoming Perform trigger.

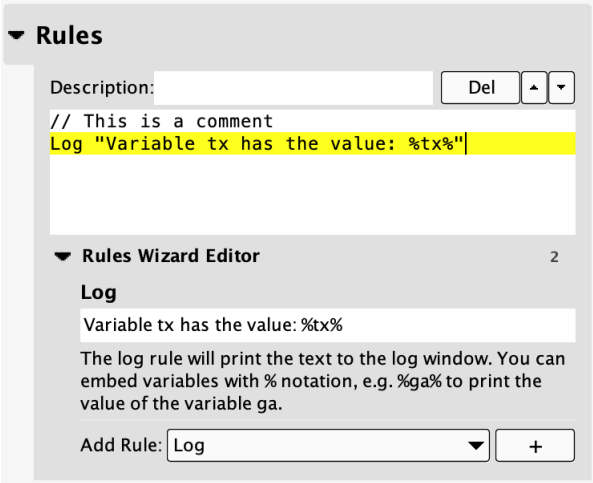


See the chapter 9.6 Perform Action for more information on Perform.

10.2.6 Log

This rule will output a line to the Log Window (see Log Window on page 23).

When this project is run on a **BomeBox**, it will add the text to the BomeBox log page if the BomeBox Log is in **verbose** mode. For more info on the BomeBox, see chapter 15: MIDI Translator in Hardware: the BomeBox.



10.2.7 Exit Rules and execute Outgoing Action

This is a direct action. Upon processing this rule, the Translator will immediately stop processing the rules and execute the Translator's Outgoing Action. These rules are commonly found coupled with Conditional rules and Labels to create complex processing statements.

10.2.8 Exit Rules and ignore Outgoing Action

This is a direct action. This rule will immediately stop processing the rules set, but will NOT execute the outgoing action. This is useful for making Translators that ONLY execute when certain conditions are met.

10.2.9 Conditional / If

Examples:

```
IF pp = 10 THEN qq=11
IF gc >= xx THEN ...
```

Conditional rules allow you to specify that a rule will ONLY execute if certain conditions are met. Conditional rules are constructed as follows:

```
IF   value/variable   ==/!=/>=/<=/>/<   value/variable   THEN
```

If the preceding conditional is true, then one of the following actions is performed:

- Assignment
- Expression
- Goto
- Perform

- Log
- Skip Next Rule
- Skip Next 2 Rules
- Exit Rules, Execute Outgoing Action
- Exit Rules, Skip Outgoing Action

Conditional Rule Operators:

`==`	EQUALS	(true example: `IF 10 == 10 THEN`)
`!=`	DOES NOT EQUAL	(true example: `IF 10 != 45 THEN`)
`>=`	GREATER THAN OR EQUAL TO	(true example: `IF 86 >= 45 THEN`)
`<=`	LESS THAN OR EQUAL TO	(true example: `IF 34 <= 34 THEN`)
`>`	GREATER THAN	(true example: `IF 10 > 4 THEN`)
`<`	LESS THAN	(true example: `IF 24 < 80 THEN`)

10.2.10 Else

Since MIDI Translator Pro 1.9.1 and BomeBox firmware 1.5.3, you can use the ELSE statement in the rules. The ELSE rule is only executed if the Rule before it is an IF statement, and only if that IF statement evaluated to false.

For example:

```
IF gt==10 THEN Log "We have a ten."
ELSE Log "Some other value."
```

You can also combine ELSE with another IF:

```
IF pp<32 THEN Log "Low"
ELSE IF pp<64 THEN Log "Low Middle"
ELSE IF pp<96 THEN Log "High Middle"
ELSE Log "High"
```

10.3 Types of Variables

There are two main types of variables in Bome MIDI Translator: *Local Variables* and *Global Variables*. Variables can be set either with incoming actions or with rules.

The life time of a Local Variable is bound to an incoming event. A Global Variable, however, will retain it's value as long as the project in Bome MIDI Translator is running.

10.3.1 Local Variables

There are 10 Local Variables defined. They are double-letter variables with equal letters: `oo`, `pp`, `qq`, `rr`, `ss`, `tt`, `uu`, `vv`, `ww`, `xx`.

Local variables retain their value as long as a given input event is being processed. Once an incoming event processing is done, the local variable is undefined. Because MIDI Translator can execute multiple incoming events simultaneously, there are then multiple sets of the same local variable. Each simultaneous translator will work on the local variable that is assigned to the respective incoming event. So local variables are an easy way to ensure that incoming events do not mess up processing of other simultaneous events which use the same local variables.

Local variables are normally the most commonly used variables, and are useful for holding temporary values. Local variables can be used in incoming actions to pass, for example, a continuous controller value to the Rules section of a Translator, where it can then be processed and resent to the outgoing action.

Note that Local Variables are not pre-initialized: if you don't define them in the Incoming Action, and you don't set them to a value in a Rule, a Local Variable can have any (random) value.

10.3.2 Global Variables

Global variables are defined by two-character combinations in the following ranges: `ga` - `gz` / `g0` - `g9` , `ha` - `hz` / `h0` - `h9` , ..., `za` - `z9` .

The global variables do not include the 10 local variables (see above)!

Examples for global variables: `h4` , `kd` , `j0` , `nb` , `zg` , etc.

> Note: MIDI Translator Pro version 1.9.0 introduced the global variables in the range `oa` - `x9` .

Global variables retain their value for the life time of the project. Global variables are useful for passing information between translators, and for remembering state.

One common use of global variables is to create a 'Shift' button on your controller, which can then control which translators are processed depending on the state of the shift control.

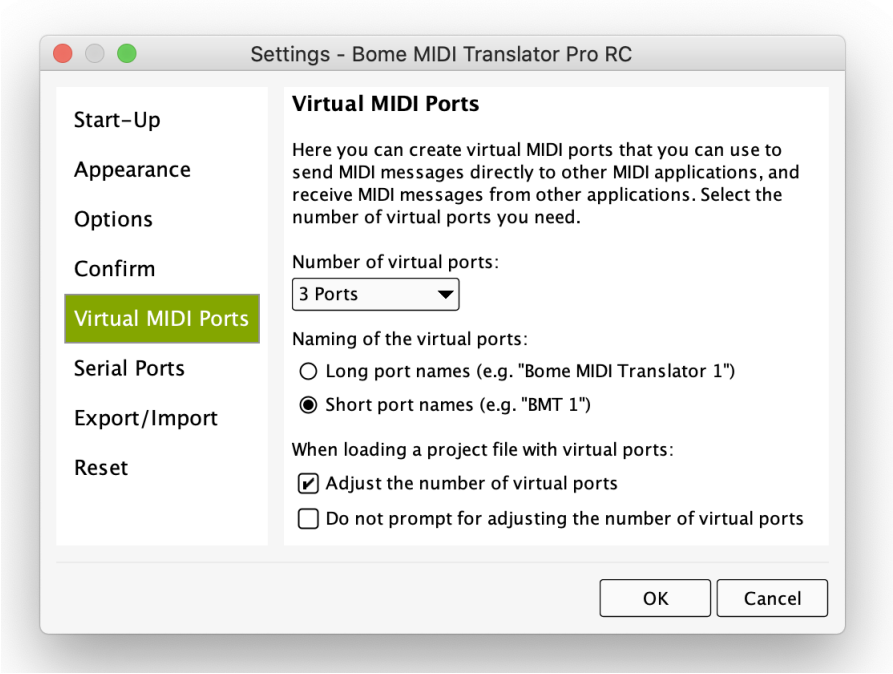
Another common use of global variables is to 'Hold' a controller's value while a timer is running, allowing you to re-send that value when the timer is done processing.

At Project start, all global variables are initialized with 0.

10.4 Using Rules and Variables

One of the most useful ways to use Rules and Variables in your Translator is the translation of a velocity or cc value to another value. Variables may be utilized in the mapping of an incoming MIDI action in a Translator by changing the last value to a variable setting instead of a static value. Variables may be used in both incoming and outgoing translator actions, allowing values input into translators to be processed, and then sent on to the outgoing MIDI port while retaining full routing flexibility.

11 Settings



Settings for Bome MIDI Translator allow the end user to modify the general behavior of the program.

The settings are stored on your computer and are not part of the project file.

Setup options for Bome MIDI Translator are divided up into these general categories:

- Start-Up
- Appearance
- Options
- Confirm
- Virtual MIDI Ports
- Serial Ports
- Export/Import
- Reset

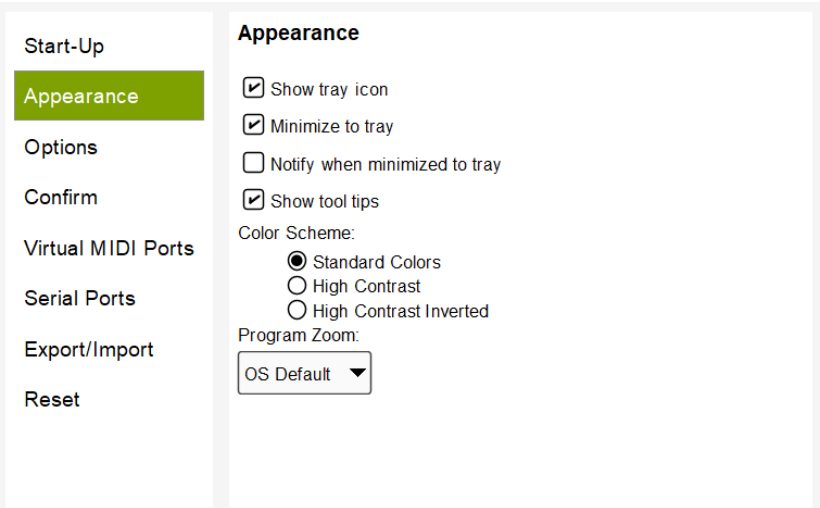
11.1 Startup Options

The Startup section deals with how Bome MIDI Translator initially starts. Having the program start up automatically with the operating system can be useful once you have all your Translators created.

- **Start minimized:**
  If checked, MIDI Translator Pro will start in minimized form, i.e. it will not show the main window. You need to (double-)click the program icon to show MT's window.
- **Show splash screen:**
  Enable or disable the showing of the program splash screen when MIDI Translator is started.
- **Auto-Start:**
  This option causes MIDI Translator to automatically start when your computer is booted.
- **Only allow one instance:**
  If this option is checked (recommended), starting MIDI Translator Pro while it is already running (e.g. hidden in the tray/menubar) will instead activate that already running instance. You can also use this to pass command line parameters to the running instance, effectively remote controlling a running instance.

11.2 Appearance

The Appearance section of the settings window deals with how the program behaves.



- **Show tray/menu bar icon**
  If enabled, a small MIDI Translator icon is visible in the tray (Windows) or menu bar (macOS). You can use it to quickly open MT's window, or to use it instead of the main icon in task bar / dock.

- **Minimize to tray/menu bar** (Windows only)
  This option is available on Windows only. If checked, and the tray icon is visible (see previous option), minimizing the program will "hide" it in the tray. The main program icon in the task bar is hidden then. Double-click the tray icon or use the icon menu to show MIDI Translator Pro again.
- **Notify when minimized to tray / menu bar**
  If this option is checked, minimizing to tray/menu bar will pop up a balloon window to tell you where MIDI Translator Pro had been minimized to – i.e. the tray / menu bar icon. After minimizing 3 times to the tray, this option is activated automatically.
- **Show tool tips**
  If this option is enabled (default), most buttons and other program elements show a small explanation when hovering the mouse over it. You can disable this behavior by unchecking this option.
- **Color Scheme**
  By default, the Standard Colors are selected. For users with eye problems or other conditions, you can select High Contrast for Black on White look of the program. Or High Contrast Inverted for a black program with white text.
- **Display icons for keystroke modifiers (macOS only)**
  If this option is enabled (default), keystroke modifiers are displayed with their macOS icons:

  Otherwise, they are displayed as on Windows with the spelled out modifier names:

- **Program Zoom**
  Here you can define a global zoom of the entire program interface. The default is "OS Default", so it uses a High DPI / retina value as defined in the OS display settings. If you like, you can override it with different zoom levels from 100% to 300%.

11.3 Options

In the Options section, general other options can be set.

- **Suppress outgoing keystroke when focused**
  If checked (default), outgoing keystrokes are only emulated if MIDI Translator is not the currently active application. This is to prevent accidental activation of functions in MIDI Translator.

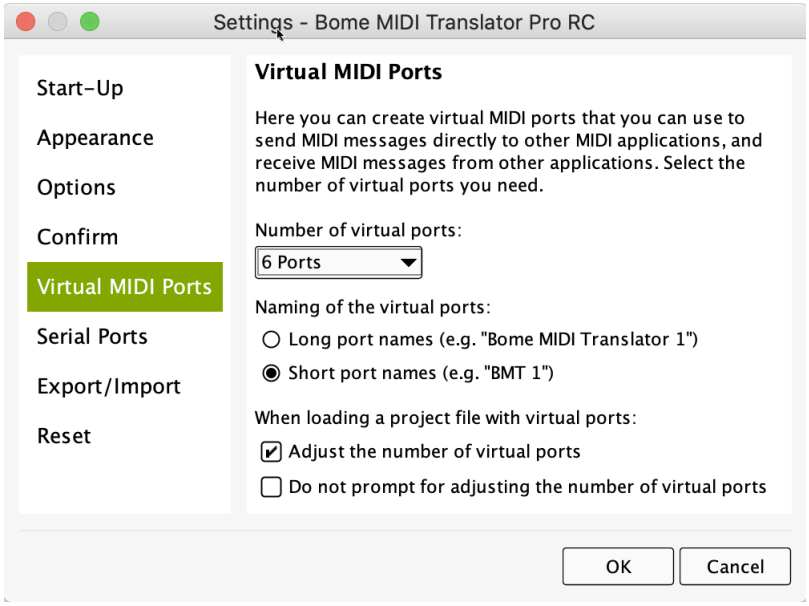
- **Ignore incoming keystrokes when focused**
  If checked, incoming keystrokes will not trigger corresponding incoming actions if MIDI Translator is not the currently active application. This is to prevent accidental activation of functions in MIDI Translator.
- **Create backup files**
  If this option is checked (default), MIDI Translator keeps saving backup files.
- **Display of middle C (MIDI note number 60)**
  There are different interpretations of which octave is the note number 60. Here you can choose which one. If unsure, leave the default C4.

11.4 Confirm

Many actions in MIDI Translator will prompt you for confirmation. They usually come with a checkbox “do not show again”. Once disabled, you can enable such confirmation prompts again by checking them here.

11.5 Virtual MIDI Ports

Bome MIDI Translator includes built-in virtual MIDI port drivers that enable the end user to send to and receive from other applications. This way you can seamlessly link up MIDI Translator as “man in the middle” between a MIDI device and an application without using any 3rd party software. Up to nine sets of virtual MIDI ports may be installed at any one time, allowing expanded control and flexibility of your MIDI routing.



Number of Virtual Ports

In the drop-down list, simply choose how many pairs of virtual MIDI ports you require in your processing (1 virtual MIDI port = 1 MIDI IN, 1 MIDI OUT), select Apply and follow the on-screen hardware installation instructions similar to the product installation covered in the Quickstart guide.

Naming of Virtual Ports

There are two different ways for how these virtual ports appear in other software on the same computer:

Long Names

This is the default, port names are named using the scheme “Bome MIDI Translator 1”.

Short Names

When selected, the virtual ports appear as “BMT 1” in other applications. This is particularly useful for applications which only display a small text field for the name of a MIDI port.

Adjusting Virtual Ports when loading Project File

Adjust the number of Virtual Ports

If this option is selected (the default), MIDI Translator will make sure that the number of virtual ports is adjusted when loading a project file. For example, if the project file uses the port “Bome MIDI Translator 3 Virtual Out”, or the alias “Bome Virtual Port 3”, it will make sure that the number of virtual ports is at least 3.

Do not prompt for adjusting the number of virtual ports

If this is checked, MIDI Translator Pro will ask you if it should adjust the number of virtual MIDI ports when loading a project file, and the currently selected number of virtual ports is not sufficient.

11.6 Serial Port Settings

In the Serial Port settings, you can configure the serial ports and serial port aliases. Note that the serial port settings are stored in the **project file**.

11.7 Export/Import Settings

11.7.1 Overview

The MIDI Translator settings Export/Import feature allows a user to backup their settings to a .bmts file to restore at a later time.

Note that Project Default Ports, Author Info, and MIDI Router connections are saved in your project files.

11.7.2 Export Settings to .bmts file

In the Options menu, use the "Export Settings" function. It allows you to export your settings to a .bmts file. These settings include:

- Window size and position
- Selected translator preset
- All program settings and preferences from the Settings Window.

11.7.3 Manually Import .bmts file

Use the "Import Settings" function in the Options menu and specify a previously saved .bmts file. This is particularly helpful if you want to transfer your settings to another computer, give them to a friend, or you have to reinstall your OS.

11.7.4 Use Command Line for Importing Settings

You can use the `/settings` command line switch to use a particular .bmts file when starting MIDI Translator. You can create a small script file that launches MT with a particular settings file (and possibly project file using the */project* switch) for quick access to a particular configuration. On Windows, script files will be a .bat or .cmd file, on macOS, you can use shell scripts (.sh) or Apple Scripts.

Note: when you quit MIDI Translator, it will write back the current settings to the settings file you specified on the command line.

11.7.5 Auto-Load of .bmts File at Start-up

There is a trick to let MIDI Translator auto-load a .bmts file: rename it the same as the app name and put it into the same folder.

On Windows, if the application executable file is `MIDITranslator.exe`, rename the settings file to `MIDITranslator.bmts`. and copy it side-by side to the same folder. Note that by default, Windows will not show the `.exe` extension.

On macOS, if the app file is `Bome MIDI Translator Pro.app`. Rename the settings file to `Bome MIDI Translator Pro.bmts` and copy it side-by side to the same folder. Note that the Finder will usually not show the `.app` extension.

This is particularly useful when running MIDI Translator from a USB thumb drive.

11.8 Reset

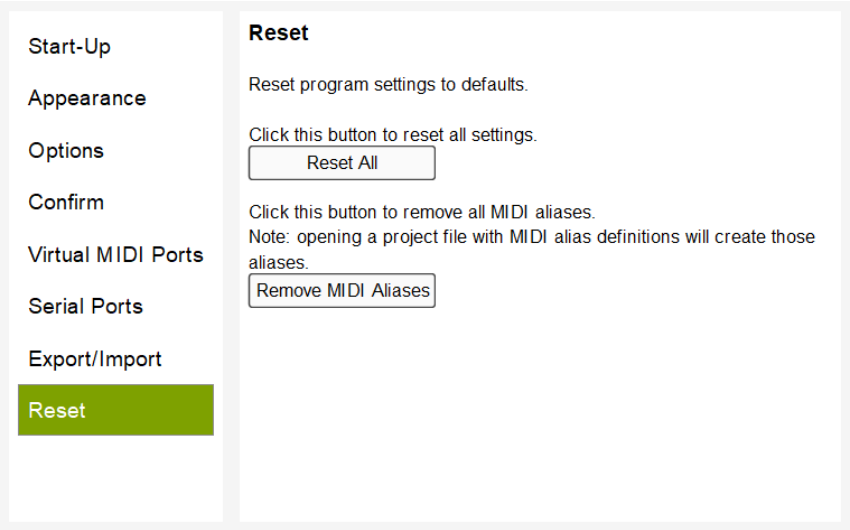
This settings screen lets you reset certain settings of MIDI Translator.

11.8.1 Reset All

Clicking this button will reset all settings to their defaults.

11.8.2 Remove MIDI Aliases

This button will remove all MIDI aliases and their associations. Note that loading a project file with stored MIDI port settings will recreate those MIDI aliases in the project file.



12 Behind the Scenes

In this chapter, the inner workings of the translation engine are explained.

12.1 Incoming Event Processing

MIDI Translator is entirely event driven: nothing happens until an event comes in. Only when an event enters MIDI Translator's engine will it become alive. An example for an event is a MIDI message coming in on a MIDI INPUT port.

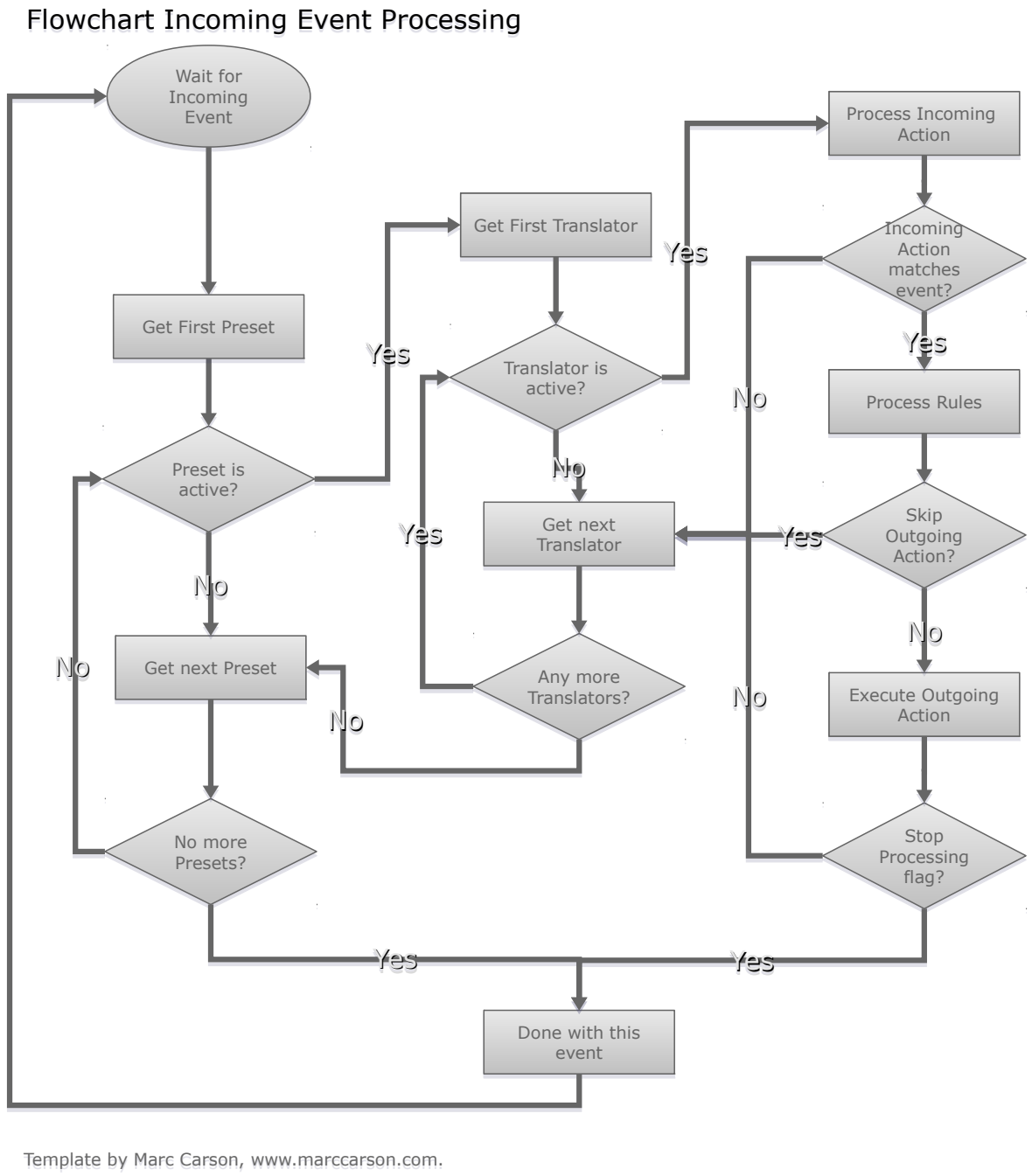
The engine looks at the first preset. If the preset is active (i.e. it is checked in the preset list), it looks at the first translator in the preset. If the translator is active and has the same Incoming Action type as the incoming event (e.g. MIDI), it processes the Incoming Action.

Processing the Incoming Action is mainly a check to see if the event matches the definitions in the Incoming Action, e.g. to compare the MIDI message in the Action with the MIDI message of the event. Some Incoming Actions may also set local or global variables (like the MIDI Incoming Action).

If Incoming Action Processing results in a match, the Rules of this Translator are executed (if any). The Rules can abort processing of this Translator. If they don't, then the Outgoing Action is executed.

When the Outgoing Action is executed, the engine looks at this translator's “Stop Processing” flag. If it is set, processing of this event is done.

In all other cases, the engine looks at the next translator and processes it exactly in the same way as the first translator: the entire process is repeated for every translator in the first preset, and then for all following presets the same way it was done for the first preset.



12.2 Executing the Outgoing Action

Once the Incoming Action processing logic (see previous chapter) determines that the Outgoing Action needs to be executed, the engine will either execute the Outgoing Action directly, or enqueue the action for asynchronous execution.

The decision if an action is executed directly or asynchronously depends on many factors out of scope of this document, but in general fastest processing of the engine in general is favored. For example, Outgoing Keystroke emulation takes a while to execute. So if you generate keystrokes from a MIDI message but also do a lot of MIDI-to-MIDI

translation, the emulated keystroke would interrupt the MIDI translations until the keystrokes are sent. For that reason, Outgoing Keystrokes are usually enqueued for asynchronous execution.

Asynchronous execution has the advantage that it allows for much better runtime performance. But on the other hand, it will be impossible to foresee in which sequence Outgoing Actions are going to be executed: in the example above, MIDI events arriving after the Outgoing Keystroke action was enqueued are likely be processed before the Outgoing Keystroke(s) are fully sent.

To make processing deterministic, the engine always guarantees that outgoing actions of the same type are executed in the correct order.

12.3 Parallel Processing

MIDI Translator's engine is using advanced optimization techniques to provide the best realtime processing performance even at heavy load and when processing many different types of incoming events.

In particular, the engine is multi-threaded, using all processor cores on modern multi-core processors. For example, the engine can process MIDI events from different MIDI ports simultaneously. Some Outgoing Actions can be executed simultaneously with other Outgoing Actions, some Outgoing Actions are forced to be executed asynchronously in a separate thread (see the example with Keystrokes in the previous chapter).

Multi-threading is nice for performance, but it may bring unexpected behavior. In particular, be aware of parallel processing when using global variables. Race conditions may occur where two events are processed simultaneously and both change the same global variable. As a general rule, you should use local variables whenever possible. Local variables' scope is restricted to the particular incoming event, so it is not possible for a concurrent event to cause problems with local variables.

13 Tips & Tricks

13.1 Make Backups!

Creating and refining MIDI Translator projects can be a lot of work. Many people only realize that when it is too late...

It cannot be said often enough: hard disks will die eventually, so regularly save your project files on an external hard drive, USB thumb drive, or network drive. At best, use a tool that seamlessly does that for you. macOS users can conveniently use Time Machine for automatic backups.

13.2 Quick Access to Different Configs

If you frequently alternate between two or a couple of different project files and would like to quickly access them, a quick&easy way is to create multiple copies of the MIDI Translator executable app file:

1. Locate the folder where MIDI Translator is installed
   (Windows, e.g.: `C:\Program Files\Bome MIDI Translator Pro`, or the `Applications` folder on macOS)
2. Select the executable app (`MIDITranslator.exe` / `Bome MIDI Translator Pro.app` – where `.exe` and `.app` aren't usually shown by Explorer/Finder)
3. Copy and paste the file:
   Windows: `Ctrl-C` and then `Ctrl-V`
   macOS: `Command-C` and then `Command-V`
4. Rename the copy to your liking, e.g. `BMT Ableton`

The trick is that from now on, the copied app will use its own settings, i.e. which project file to load, location of the program window, which preferences and settings, MIDI ports, etc.

Alternatively, you can create start scripts (.bat/.cmd on Windows, .sh or AppleScript on macOS) which use the command line switches to select a settings file and/or project file.

13.3 Running from a USB Thumb Drive

Many users need to be mobile and sometimes need to run their setup on other computers, or have a backup solution available.

13.3.1 Windows

You can copy the entire installation folder to a thumb drive. It is usually found in: `C:\Program Files\Bome MIDI Translator Pro`

If you need virtual MIDI ports, we recommend that you put the installer on the thumb drive. There is no other convenient way to install the virtual ports other than to run the installer of MIDI Translator.

Once the virtual ports are installed, however, you can run MIDI Translator from a thumb drive.

13.3.2 macOS

Installing MIDI Translator on an external disk or a USB thumb drive is not a big problem for the macOS version. Just copy the application from your Applications folder to the thumb drive.

13.3.3 Auto-load Settings

In order to use the same settings when running from the thumb drive as you use on your computer, do the following:

1. Run MIDI Translator, and use the “Options|Export Settings” menu to export your current settings to a .bmts file.
2. Save the file under the same name as the MIDI Translator app, but with the .bmts extension, e.g. `Bome MIDI Translator Pro.bmts`.

Now when you start MIDI Translator from the thumb drive, it'll find the same-named .bmts file and use that instead of one installed on the computer.

13.3.4 Auto-load Project

In order to auto-load a project file, use the same trick as above for the settings file: save your project file on the thumb drive under the same name as the app name, but keep the .bmtp file name extension. Like that, MIDI Translator will load that project file (if it exists) when started from the thumb drive.

13.3.5 Execute File Action on Thumb Drive

If your project executes files or shows documents using the Execute File Action (Outgoing), using relative paths will be resolved as relative to the location of the project file. So omitting a folder path in the filename will make sure that the file is opened from the same location as the project file.

13.3.6 Variable Storage on Thumb Drive

To maintain a persistent Variable Storage on the thumb drive, configure the Variable Storage Action to use the File option and use a relative folder or no folder path at all. Then the variables are saved and loaded along side the project file.

13.3.7 Using Multiple Configurations

As shown in the tip Quick Access to Different Configs above, just add multiple copies of the executable app on the thumb drive and use a matching settings file and project file to go along.

13.3.8 Using Script Files

For the more advanced user, you can create start scripts that use the command line switches for using particular settings and project files. See also: Use Command Line for Importing Settings.

13.4 Timer with 0ms

There is a trick to immediately execute a timer: use a one-shot timer with 0ms duration. It will be processed very efficiently, usually already in parallel to the current event (see also Parallel Processing).

In previous versions of MIDI Translator Pro, 0ms Timers were also popular for triggering multiple actions at once. With current versions of MIDI Translator Pro, the Perform Action is more suitable for that.

13.5 Leverage the Perform Action

The Perform Action can be used similar as a function in programming languages.

In particular, it allows you to trigger a series of actions with one or more incoming actions:

```
Translator 0:
         Incoming: MIDI message 1
         Outgoing: Perform "My Function", 1
Translator 1:
         Incoming: MIDI message 2
         Outgoing: Perform “My Function”, 2
Translator 2:
         Incoming: Perform "My Function"
         Outgoing: out 1
Translator 3:
         Incoming: Perform "My Function"
         Outgoing: out 2
Translator 4:
         Incoming: Perform "My Function"
         Outgoing: out 3
```

That's an easy and fast way to trigger 3 outgoing actions from 2 incoming actions.

13.6 Multiple Actions in one Translator

Currently, it is not possible to execute multiple Outgoing Actions from one Translator. Here are some other ways how to achieve that:

1. The usual work-around is to duplicate the Translator and only modify the Outgoing Action. Now there are 2 Translators with the same Incoming Action, but with different Outgoing Actions. They will be executed one after another, because both Incoming Actions will be triggered from the (matching) event. Note that for this to work, you must uncheck “Stop Processing”.

2. For the MIDI outgoing action, you can use the Raw MIDI outgoing type and concatenate multiple message directly in the MIDI message field, e.g.:
   `90 40 7F 90 44 7F 90 4A 7F`
   will play 3 notes directly.

3. For a different way to trigger multiple Outgoing Actions “at once”, see Leverage the Perform Action.

13.7 Performance Optimization

For most use cases, MIDI Translator's performance will be near real time, so there should not be the need to optimize. However, for larger projects (i.e. 100's of Presets / 1000's of Translators), it will be worthwhile to know some optimization techniques.

13.7.1 Deactivate Presets

Every Translator is “processed” for every event when its owning Preset and itself are active. Now for a couple hundred Translators this will not be a problem, but you may experience a performance degradation in large projects (also depending on processor).

One easy way to optimize your project is to make sure that you disable as many Presets as possible. When a Preset is disabled, none of its Translators are even looked at during processing of an event. So it's worthwhile to group Translators that are used together into Presets and activate the Presets as needed.

For example, let's say that the first Preset is an “Initialization” preset with 100 Translators that execute after the project was opened. Subsequently, after initialization is fully done, every Incoming Event will walk through those 100 initialization Translators before reaching any useful Presets.

A straight forward way to solve this is to add a last Translator to the Initialization preset like this:

```
Preset 0: "Initialization"
Translator 0: init 1
         Incoming: On Project Opened
         Outgoing: [initialization 1]
...
Translator N: "deactivate Init preset"
         Incoming: On Project Opened
         Outgoing: Deactivate Preset "Initialization"
```

This works, and will give the performance gain. However, there is a practical problem: when working on the Project, saving it will save the “Initialization” preset in deactivated state. So the next time you load it, it won't be called, because it's not active. So you'd need to activate it manually every time before you save the project to ensure it's saved in “active” state.

One way to solve this is to add another “Project Open” preset with just one Translator. The “Project Open” preset will stay active and it will just activate the Initialization preset. The Translators in the Initialization preset are modified to react to “when this preset is activated”:

```
Preset 0: "Project Open"
Translator 0: trigger Initialization
         Incoming: On Project Opened
         Outgoing: Activate Preset "Initialization"

Preset 1: "Initialization"
Translator 0: init 1
         Incoming: current preset is activated
         Outgoing: [initialization 1]
...
Translator N: "deactivate Init preset"
         Incoming: current preset is activated
         Outgoing: Deactivate Preset "Initialization"
```

This way has another advantage: you can manually trigger the initialization anytime during development of your project by simply activating the “Initialization” preset.

13.7.2 Use “Stop Processing”

Another way to optimize processing is to bail out of event processing when all relevant Translators have processed it.

The way to do that is to identify Translators which are the only ones with this particular Incoming Action (e.g. a special MIDI message). Then check the *Stop Processing* flag so that the event will not be further tried to be processed by following Translators.

Of course, this also works if you have multiple Translators reacting on the same Incoming Action: only activate *Stop Processing* for the last of the series of Translators with the same Incoming Action.

Refer to chapter 8.2.3: Stop Processing on page 34.

13.7.3 Avoid Redundancy

When a project grows over time, often similar things are done in two different ways, or too much use of copy/paste caused duplicated Translators. Also a common thing is to forget to remove Translators that were once added for testing only. So it is a good idea to look through the Presets with an eye for duplicated functionality.

13.7.4 Use Project/Preset Default Ports

A common problem that is hard to detect is sending MIDI messages to all MIDI ports. When no default ports (in Project Properties / Preset Properties) are set, an Outgoing MIDI Action will send MIDI messages to all open ports.

As long as only one MIDI port is opened and in use, that's OK. But then, when you open multiple MIDI ports, your Outgoing Actions will suddenly send all MIDI messages to multiple MIDI ports. Often, this remains undetected, because the project still works. But the workload is much higher.

The best way to do that is to group the translators into presets by MIDI ports, and then use the Preset default ports to specify only one MIDI OUT port per Preset as default port.

13.7.5 Use the Log Window For Development

The *Log Window* is a powerful tool for watching your Project in action. Every now and then you should double check your project by verifying in the Log window that the Actions are executed and processed as planned.

You can also use the `Log` rule to output any text or variable value to the Log Window from the Rules section. For more information, see chapter 10.2.6 Log on page 101.

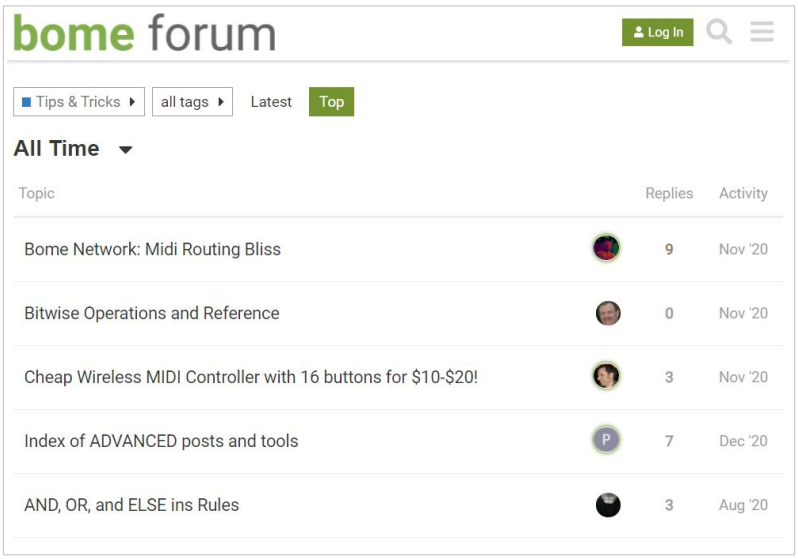
Turn off the Log Window when actually using MIDI Translator in action, to ensure best performance.

For more information, refer to chapter 4.9: Log Window on page 23.

13.8 Tips & Tricks in the Bome Forum

We maintain a growing collection of tips and tricks in the Bome Forum. Why don't you stop by and check it out?

https://forum.bome.com/c/tips-tricks/7

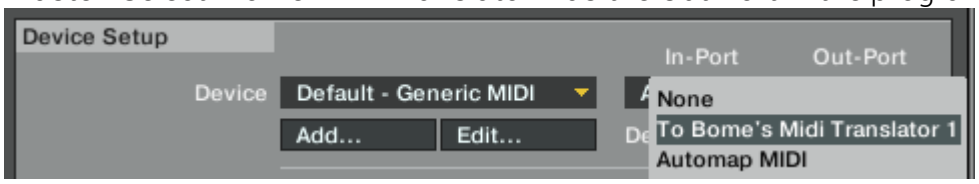


14 Usage Example

14.1 Traktor / Ableton Live Sync

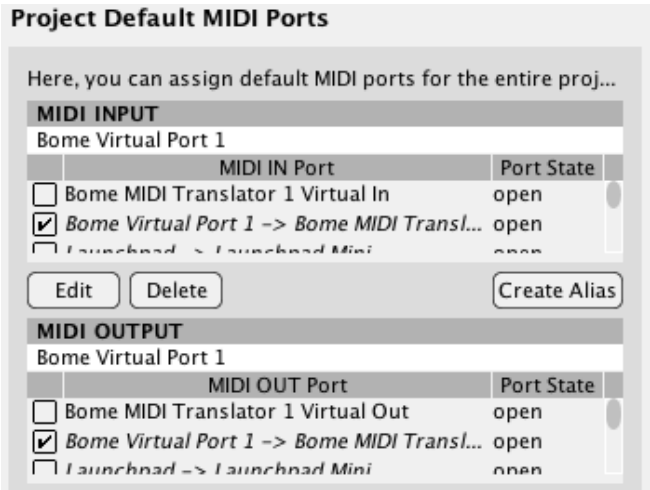
Virtual MIDI ports are available for use as unidirectional MIDI ports, so you only need one virtual MIDI port to get MIDI data into a target application like Traktor, and to get data back from it.

1. The first step is to select the Bome virtual port as the MIDI output in your MIDI clock source application. In this example we're using Traktor to act as the MIDI clock master. Select "Bome MIDI Translator 1 as the Out-Port in the program preferences.

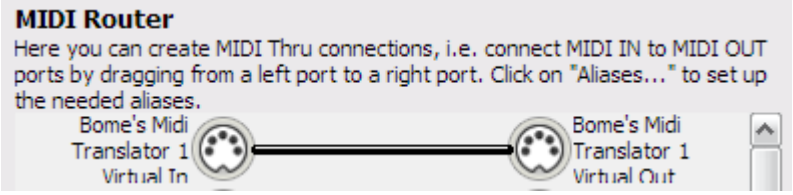


   Make sure you have the program set to transmit the clock signal by going into the MIDI Clock category of the preferences dialog and making sure "Send MIDI Clock" is checked. Also make sure that you have MIDI clock playing by accessing the metronome section of the user interface and clicking the "Play" button in the MIDI CLOCK section.

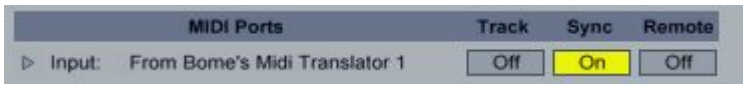
2. The next step is to configure Bome MIDI Translator to activate the virtual ports we wish to use coming IN, going OUT and then to link them via the program MIDI Router. Activate the Virtual MIDI port OUT and IN devices by going into Settings / MIDI Ports and putting a checkmark next to each MIDI device.



3. Next you'll need to make a link between the IN device and the OUT device in the MIDI Router. Open the MIDI Router by clicking on Settings / MIDI Router / Project Properties. Drag a link between Bome MIDI Translator 1 Virtual IN and Bome MIDI Translator 1 Virtual OUT. This will tell MIDI Translator to pass all data from Traktor thru to our MIDI clock destination source Ableton Live.



4. The final step is to activate our MIDI clock control in our destination application. In Live's preferences dialog window, select the MIDI Sync category and turn Sync on for "Input: From Bome MIDI Translator 1" to tell Live to receive MIDI clock from MIDI Translator.



15 MIDI Translator in Hardware: the BomeBox

Many users of Bome MIDI Translator Pro need to use the software in environments where using a computer is not convenient, or even dangerous: on stage, on tour, or in a recording room of a studio (fan noise!).

For that, we've developed the BomeBox:

https://www.bome.com/products/bomebox

The BomeBox has MIDI connectivity via

- USB (host, up to 32 devices via USB hub)
- MIDI-DIN in and out
- Ethernet (Cat-5)
- WiFi

And, best of all, the BomeBox **runs all your MIDI Translator Pro project files**, standalone, without a computer.

You can even use Incoming Keystroke actions (by connecting a QWERTY keyboard via USB / wireless dongle), and serial port input/output (by connecting serial port adapter(s) to the USB connector.

You can find out more about the BomeBox, including where and how to purchase it, here:

https://www.bome.com/products/bomebox

16 Reference

16.1 Terminology

- **Incoming Action:** An Incoming Action (or Incoming Trigger) is the definition that causes an associated Translator to start. Incoming Actions can be e.g. MIDI message, Keystroke, Timer interval elapsed, Preset Changes, ...
- **Name:** The Name refers to the name given to an individual Translator or Preset.
- **Outgoing Action:** An Outgoing Action is the definition of the action to execute when a translator's Incoming Action is triggered. An Outgoing Action event can be for example a MIDI message sent to a MIDI port, an emulated keystroke, starting a Timer, etc.
- **Preset:** A Preset is a collection of Translators. There are individually named Presets, which may be switched and activated separately.
- **Project:** A Project is a collection of Presets, grouped by a set of common attributes such as MIDI in/out/thru settings, appearance settings, and other common application-wide attributes.
- **Timer:** A Timer is an internal function of Bome MIDI Translator that will generate (repeated) events that can be processed by way of a Translator with the Timer Incoming Action.
- **Translator:** A Translator is an individual "rule" defining the translation of a single 'Incoming Action' event through to a single 'Outgoing Action' event.

16.2 Keyboard Shortcuts

On **macOS**, the shortcuts are used with the Command key instead of the Ctrl key.

Global	
`Shift+Esc`	MIDI Panic - Stops all MIDI information immediately
`F1`	Help – Show this user's manual
`Ctrl-O`	Open Project - Opens an existing MIDI Translator project file
`Ctrl-S`	Save Project - Saves current project, prompting for a name if first save
`F12`	Save Project As... - Saves current project as a new file name
`Ctrl-F4`	Close Project - Closes current project, prompting for save if applicable
`Shift-Ctrl-R`	Restart Project
`Alt-F4` / `Command-Q`	Quit MIDI Translator Pro (Exit)
`Command-H`	Hide MIDI Translator Pro (macOS only)
`Ctrl-N`	select next translator
`Ctrl-M`	select previous translator
`Shift-Ctrl-N`	select next preset
`Shift-Ctrl-M`	select previous preset
`F5`	show or hide the Properties Sidebar
`F6` / `Alt-` `	cycle project/preset/translator/properties
`F7`	focus Properties Sidebar
`F8`	show/hide Log Window
`Ctrl-F8`	focus Log Window (and show if not already visible)
`Ctrl-0`	focus project entry
`Ctrl-1`	focus preset list
`Ctrl-2`	focus translator list
`Ctrl-3`	go to project properties
`Ctrl-4`	go to preset properties
`Ctrl-5`	go to translator properties / general
`Ctrl-6`	go to translator incoming action
`Ctrl-7`	go to translator rules section
`Ctrl-8`	go to translator outgoing action
`Shift-Ctrl-T`	Add Translator entry
`Shift-Ctrl-P`	Add Preset

Lists	
`Ctrl+C`	Copy to clipboard
`Ctrl+V`	Paste from clipboard
`Ctrl+X`	Cut to clipboard
`Ctrl+D`	Duplicate selected entry (preset or translator)
`Del` / `Backspace`	Delete currently selected entry
`F2`	Rename currently selected entry
`Ctrl+A`	Select All entries in active list
`Ctrl+Up` / `Ctrl+Down`	move selected entry (or entries) up/down
`Shift-Ctrl-Up`	Move selected entry to top
`Shift-Ctrl-Down`	Move selected entry (or entries) to bottom
`ENTER`	Go to Properties Sidebar

Translator Editor	
`Alt+C`	MIDI Capture (for MIDI as Incoming or Outgoing Action)

16.3 Command Line Switches

The following commands can be used on the command line when launching Bome MIDI Translator Pro. If only one instance is enabled (see Startup Settings), you can also control a running instance.

General Operation

`/debug`
`/info`
`/error`
`/timedebug`
	Activate debugging output (some of which only on command line)

`/silent`	No debug output.
`/nodebug` `/nosilent`	Reverse the meaning of the debug flags.
`/multiinstance`	If this parameter is specified, a new instance of the program will be started, irrespective if another instance is already running.
`/settings <filename>`	Use the given .bmts file instead of using the last session's settings. At program exit, the settings will be written to that file.
`/noShow`	Do not activate and show main window if it is currently hidden or minimized.
`/close`	Carry out the command line parameters and then exit (do not start the MIDI Translator window).

Translation Project

`/project <filename>`	Load the given .bmtp file at startup.
`/addProject <filename>`	Add the presets in the given project file to the current project. You can specify additional `/addproject` commands to merge multiple project files.
`/closeProject`	If a project is currently loaded, close it.
`/setVar <var> <value>`	Set the value of a (global) variable. E.g.: `/setVar gc 1234 /setVar h1 0xABEF`

Since version 1.9.0.

`/triggerTimer <name>`

Fire the named timer once as if its interval just elapsed. All incoming action with Timer type and with the given timer name will be triggered.
Since version 1.9.0.

`/triggerPerform <name>,param1,param2,...`

Trigger the named Perform action(s). The parameters are optional. You should embrace name and parameters in quotes to make it one argument. E.g.:
`/triggerPerform "My Function,23, g0"`

MIDI commands

`/rescanMIDI`

Refresh the list of MIDI devices. This command is usually not necessary, because MIDI Translator Pro automatically updates the list of MIDI devices.

`/assignInAlias <Alias> <Device>`

Assign a MIDI Input Alias to the given MIDI input device. If the given Alias name does not exist, it will be created. If the device does not exist, this command will not succeed. To unassign an alias, specify an empty device: `/assignInAlias "MyAlias" ""`.
Since version 1.9.1.

`/assignOutAlias <Alias> <Device>`

Same as `/assignInAlias`, but for MIDI Output aliases/devices.
Since version 1.9.1.

`/delInAlias <Alias>`

`/delOutAlias <Alias>`

Remove the given MIDI input/output Alias. This command will not succeed if the given alias is currently in use. Also, auto-aliases cannot be removed.
Since version 1.9.1.

`/addRoute <Source> <Destination>`

Add a MIDI Route from the given Source to the given Destination. Source and Destination can be a MIDI Alias or a MIDI Device. By default, Source must be a MIDI Input alias/device, and Destination must be a MIDI Output alias/device.

For adding advanced MIDI routes, use the `<` prefix to qualify an alias/device as MIDI Input, and the `>` prefix to qualify an alias/device as MIDI Output.

Examples:
`-addRoute "<My MIDI IN" ">MyMIDI OUT"`

	Default case: MIDI IN to MIDI OUT
	`-addRoute ">UM1" "<UM1"`
	Advanced: MIDI OUT to MIDI IN (loop-back)
	`-addRoute "<Launchpad" "<My Clone"`
	Advanced: IN to IN (input duplication)
	`-addRoute ">BMT 1" ">BMT 1"`
	Advanced: OUT to OUT (output duplication)
	
	Since version 1.9.2.

`/delRoute <Source> <Destination>`	Remove the given route identified by Source and Destination aliases/devices. See the `/addRoute` command for more information on the alias/device specification.

`/enableRoute <Source> <Destination>`
 	
`/disableRoute <Source> <Destination>`
	Enable or disable the given route identified by Source and Destination aliases/devices. See the `/addRoute` command for more information on the alias/device specification.

`/bmidi <num ports>`	Select the number of virtual MIDI ports you wish MIDI Translator to use (0...9). Windows: If currently there are more ports installed than the given number of ports, remove ports so that only the given number of ports remain installed.
`/bmidiAtLeast <num>`	Like `/bmidi`, but never remove ports.

Serial Port commands

`/assignSerialPortAlias <Alias Name> <Port Name>`	Assign a real serial port to the given Serial Port Alias. To unassign the alias, specify an empty Port Name: `""` Since version 1.9.2.
`/delSerialPortAlias <Alias Name>`	Remove the given Serial Port Alias. Since version 1.9.2.
`/delSerialPortDevice <Port Name>`	Remove the given Serial Port. This only works for pending devices. If the given serial port exists (is plugged in), this command fails. Since version 1.9.2.
`/setSerialPortBaudrate <Port Name> <baudrate>`	Set the baud rate (in bits/second) for the given serial port. For Port Name, a serial port device or a serial port

	alias can be specified. Since version 1.9.2.
`/addSerialMIDI <Port Name> <MIDI Name>`	Add a serial MIDI device: the given serial port (device or alias) is mapped to an internal MIDI port. The MIDI Name is the name of the internal MIDI port. The serial port must exist for this command to succeed. Since version 1.9.2.
`/delSerialMIDI <Port Name> <MIDI Name>`	Remove a serial MIDI device. Since version 1.9.2.

16.4 Menu Reference

File

- New: Close the current project and start a fresh one.
- Open: Open an existing MIDI Translator project file.
- Save: Save current project, prompting for a name if first save.
- Save As: Saves current project with a different file name.
- Export Project as Text…: create a human readable text file with all your presets and translators.
  Note: you cannot read this file back into MIDI Translator!
- Close Project: Close the current project, prompting for save if applicable.
- Restart Project: Initialize the loaded project as if you'd just loaded it: clear global variables, and invoke the Project Opened event.
- Project Properties: Show the Project Properties in the right Properties Sidebar.
- Exit: Exit Bome MIDI Translator, prompting for save if applicable.

Edit

- Add Preset: Create a new preset in currently open program template
- Add Translator: Add new blank translator to currently active preset
- Cut: Remove currently selected preset or translator and place in program clipboard
- Copy: Copy currently selected preset or translator to program clipboard
- Paste: Create a new preset/translator identical to the one previously 'cut' or 'copied' to clipboard
- Delete: Remove currently selected preset or translator
- Duplicate: Create an identical copy of currently selected preset or translator
- Rename: Rename currently selected preset or translator
- Activate: Enable currently selected preset or translator to process incoming and outgoing events
- Deactivate: Disable currently selected preset or translator from processing incoming and outgoing events
- Move: Sub-menu - Move selected preset or translator up/down/top/bottom
- Export Preset As Text File: Save currently selected preset as a readable text file for easy display
- Properties: Show and activate the Properties Sidebar for the currently selected preset or translator.

MIDI

- Project Default Ports: Select the default MIDI ports for your project
- Preset Default Ports: Select the default MIDI ports for the currently selected preset
- Virtual MIDI Ports: open the settings screen to set up virtual MIDI ports

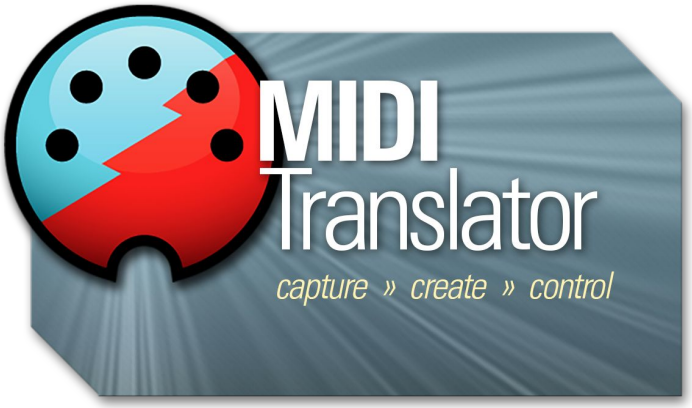
- MIDI Router: Create MIDI Thru connections between MIDI interfaces
- Rescan MIDI Ports: Rescan for open and closed MIDI ports
- Open Project Port Aliases: Open all MIDI ports which are used in the project and close all other ports
- Edit Project Port Aliases: invoke a dialog which lists all MIDI Port Aliases which are used somehow in the project. These ports are Project Default ports, all Preset Default Ports, MIDI ports selected in Incoming MIDI actions and Outgoing MIDI actions, and any ports used in the MIDI Router.
- Convert Raw MIDI to Simple MIDI: go through your project and find occurrences where you entered simple MIDI messages as raw hex MIDI. This function will convert it to normal MIDI message translators. Any more complex Raw MIDI actions are not touched.
- Panic: disable event processing and send All Sound Off to all open MIDI ports

View

- Log Window: Show program log window, which displays detailed information about incoming and outgoing actions, as well as letting you display all global system variables
- Event Monitor: Display Activity Monitor on the bottom of the program window to monitor MIDI and program activity
- Properties Sidebar: show or hide the right Properties Sidebar
- Project Properties: show the project properties in the Properties Sidebar
- Preset Properties: show the preset properties in the Properties Sidebar for the currently selected preset
- Translator sub-menu: show Translator properties in the Properties Sidebar and focus on General options, Incoming action, Rules, or Outgoing action.
- Settings: open the settings window with general options for the entire program. On macOS, the Settings are available from the left application menu.

Help

- Help: Show this user's manual
- Get Support: go to the support web page (online)
- Support forums: open the support forums in an Internet browser (online)
- Video Tutorials: go to our YouTube channel with dozens of tutorials (online)
- Check for Updates: check online for an updated version of MIDI Translator
- Purchase MIDI Translator: Open web browser page detailing purchase information (online)
- Change License: open a window where you can enter a different license key (full version only)
- Remove License Key: remove the license key from this computer (e.g. when you have temporarily installed MIDI Translator Pro on somebody else's computer)
- About: Show program license, version and copyright information. On macOS, the About screen is available from the left application menu.


MIDI
Translator
capture » create » control