bome software

everything sounds.

Bome MIDI Translator

USER MANUAL

2016-07-25

Table of Contents

1 Welcome

Thank you for choosing Bome MIDI Translator!

This document contains:

- MIDI Translator Quick Start
- MIDI Translator User's Manual

Announcement List:

- In order to get update notifications, **sign up to the announcement list**. The volume on this list is very low and after email verification, you can select which news categories you're interested in.

User Support:

- **Discussion Forums**: get professional help for your questions
- **Contact Bome Software**: support for registered users, purchase inquiries, OEM licensing, bundling, site licenses, etc.

We hope you enjoy our software,

The Bome Software Team

2 Quick Start

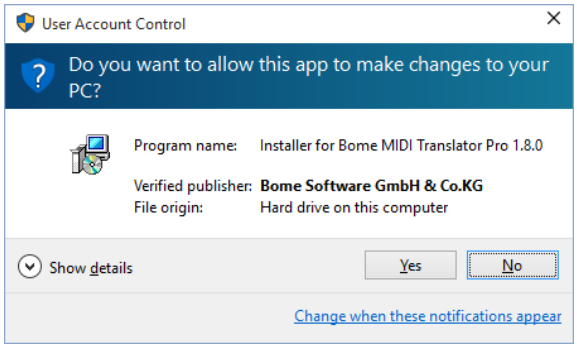
2.1 Installation on Windows

Install Bome MIDI Translator with the options that you require. If you wish to transmit MIDI messages to another application, make sure to install the Bome Virtual MIDI Port drivers during installation. On Mac computers, the virtual ports are always installed.

2.1.1 General Installation

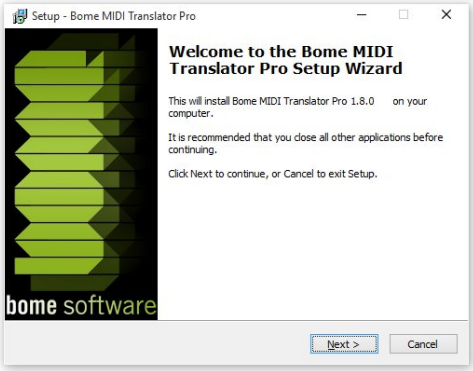
> USER ACCOUNT CONTROL

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



> INITIAL INSTALL SCREEN

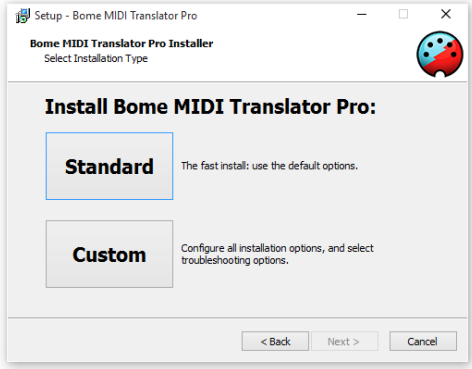
Then, the installation wizard will begin:



> INSTALLATION TYPE

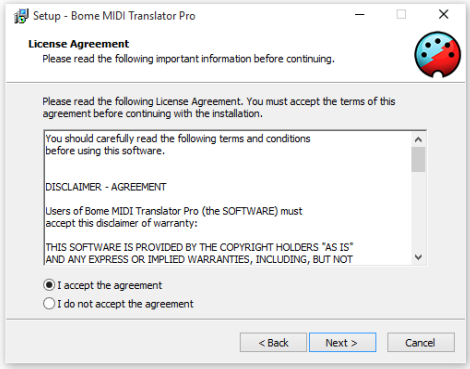
Usually, you'll choose Standard here for quick and easy installation with the common options.

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 MIDI Translator on your computer.

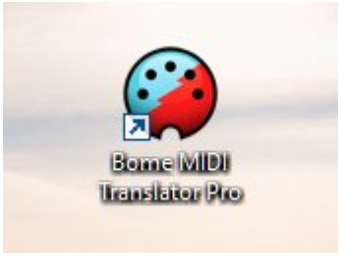
For Windows XP users, please see the following chapter for a guide through the Virtual Driver installer.

> COMPLETION

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

> STARTING MIDI TRANSLATOR

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



2.1.2 Windows XP: Virtual Port Installation

On Windows XP, the virtual port installation requires some user action (unless you have chosen to not install virtual ports in Custom Install mode). This is explained in the following paragraphs.

> Windows Vista, 7, 9, 10 will install the virtual MIDI ports without user interaction.

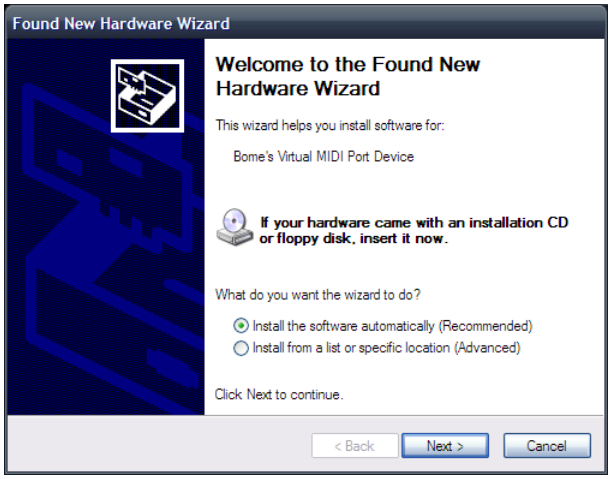
> WINDOWS XP: INSTALL WARNING

If you have chosen to install support for Bome virtual MIDI ports, you will be presented with a dialog box detailing the installation process. Read the instructions carefully and then click OK.

Windows XP: virtual port install warning

> WINDOWS XP: VIRTUAL PORT INSTALL

Bome virtual MIDI ports are installed much like a real hardware device. Click "Install the software automatically" and then click Next to continue.



Windows XP: virtual port hardware install

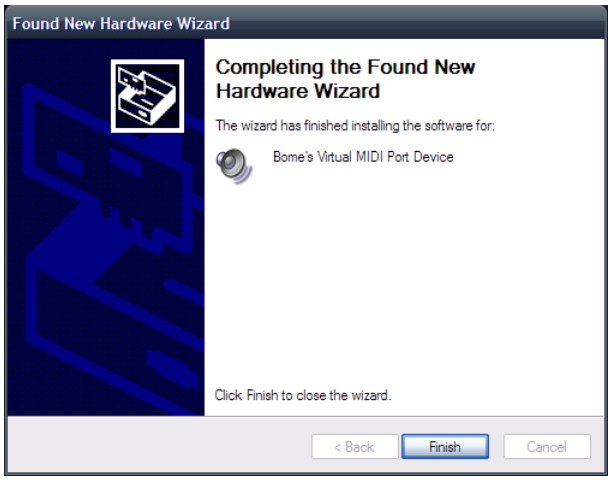
> WINDOWS XP: LOGO TESTING

Be sure to click *Continue Anyway* when you are prompted about Windows Logo Testing.

Windows XP: logo testing warning

> WINDOWS XP: VIRTUAL PORT INSTALLATION COMPLETE

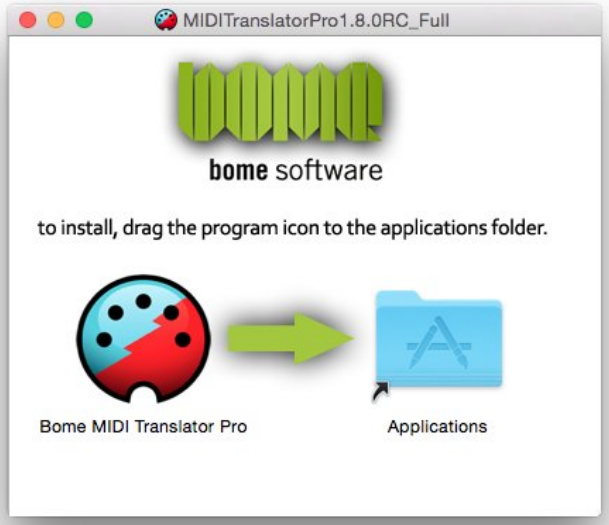
Press the Finish button to complete the installation of the virtual MIDI port.



Windows XP: virtual port installation complete

2.2 Installation on OS X

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.

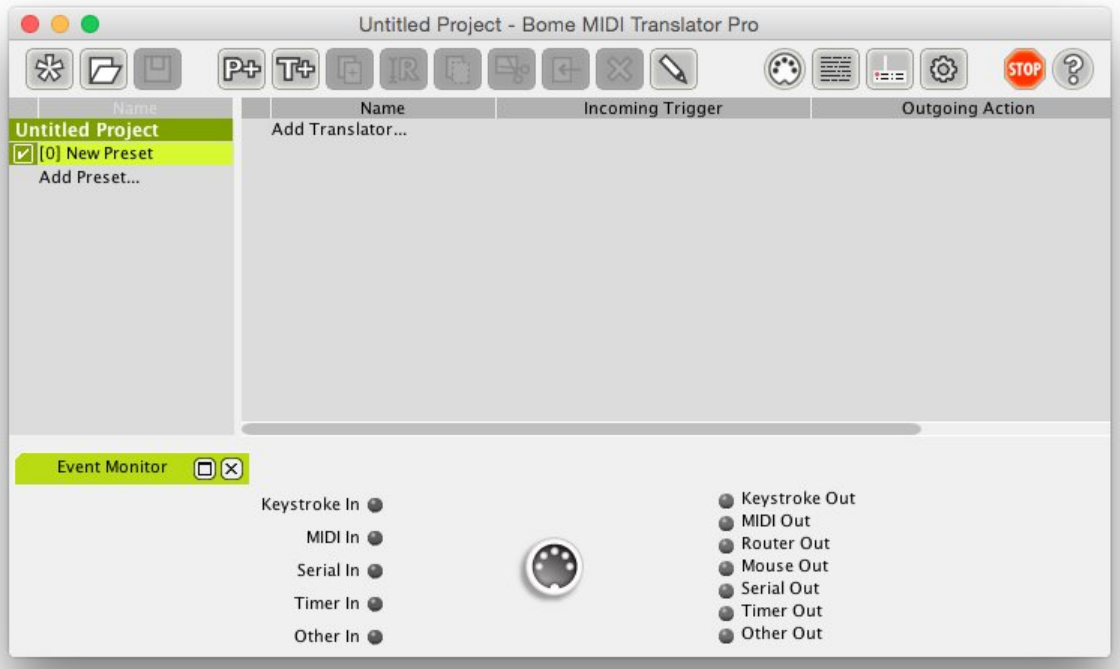


To install Bome MIDI Translator in your Applications folder, simply drag the MIDI Translator Icon to the Applications icon visible on the right in the same window. Alternatively, you can also double-click the MIDI Translator icon – it will detect that it runs off of a DMG volume and offer to install itself in the Applications folder for you.

If you want to use AppleScript with MIDI Translator, you *must* install it in the *Applications* folder. Otherwise, you can also drag the MIDI Translator Icon to a different folder or the desktop to install it there.

2.3 Start MIDI Translator

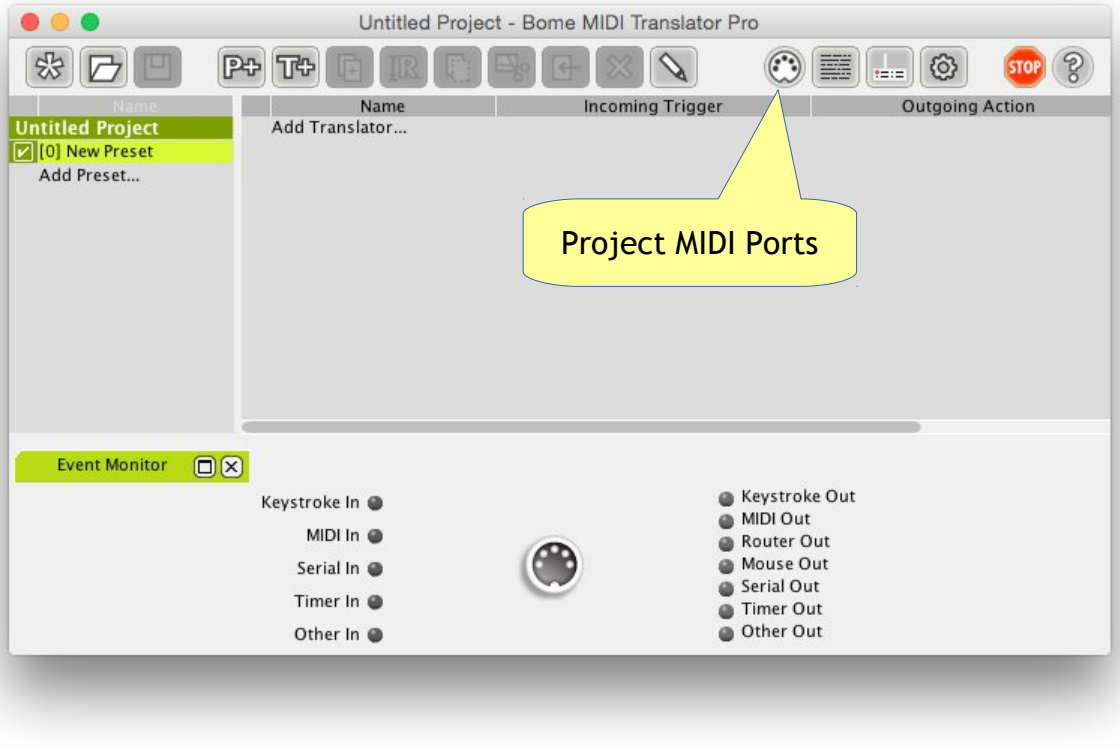
Once installed, start Bome MIDI Translator by double-clicking its icon.



2.4 MIDI Setup

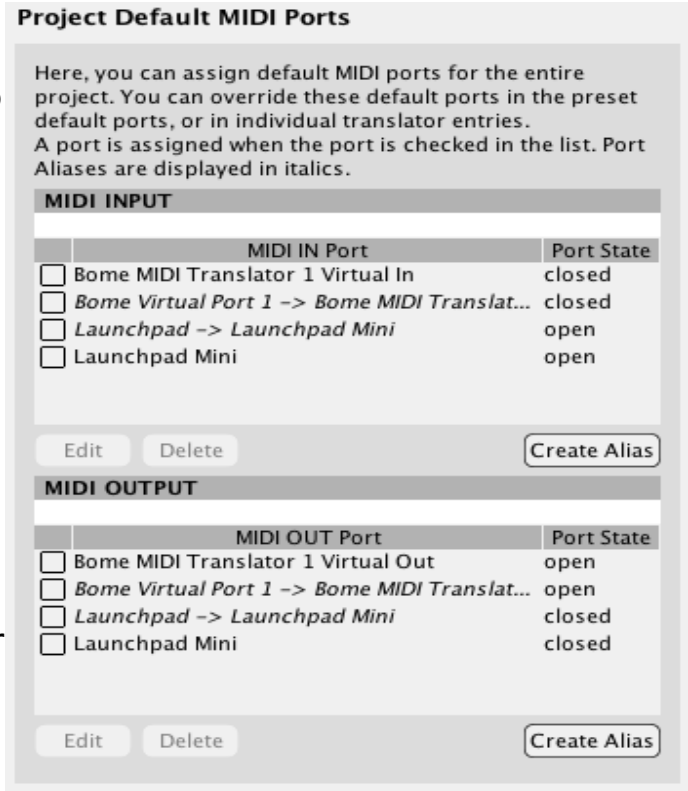
2.4.1 MIDI Settings

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") in the left pane.



MIDI settings

2.4.2 Define MIDI Ports

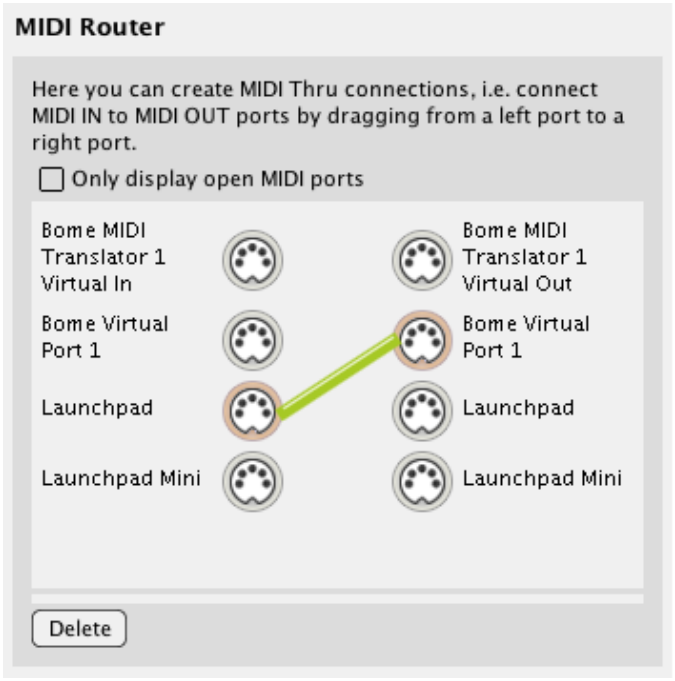


Next, specify the MIDI input and output ports you will be using. The MIDI OUT ports will be the ports to which translated MIDI messages are sent to.

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.

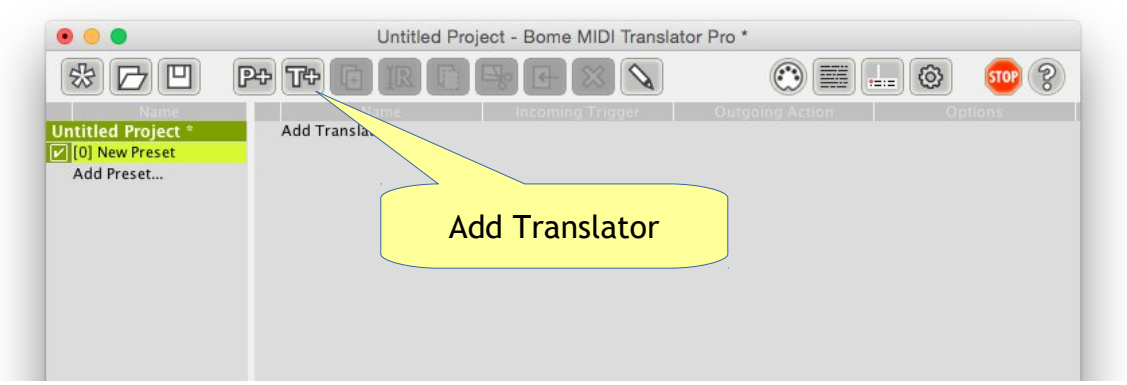
Access the MIDI Router in the Project Properties: click on the filename on top of the left list, or use the View menu (keyboard shortcut: Ctrl+3). On the right, scroll down until you see the section titled MIDI Router.

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

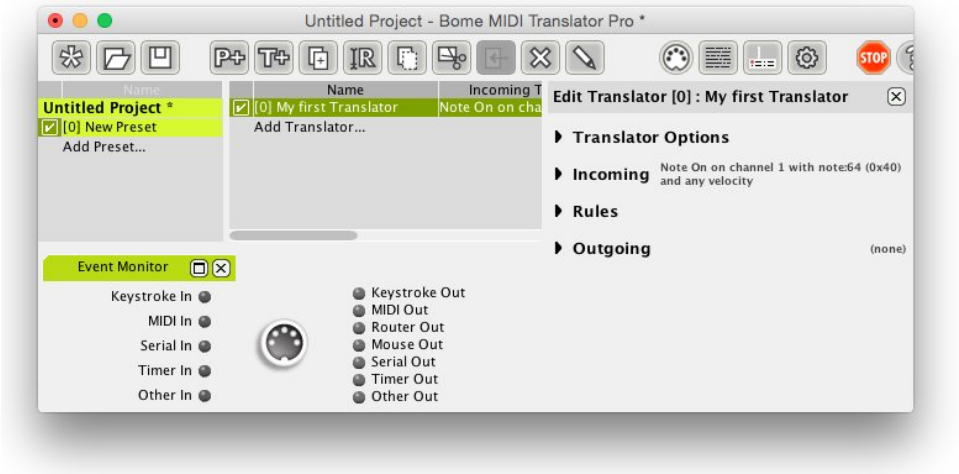
You now should have the MIDI interface settings properly configured. Test that they're working correctly by moving a controller on your MIDI device and checking to see if the corresponding light illuminates on the Event Monitor, located at the bottom left.

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 right translator properties are not visible, double click the translator object to enter the Translator Edit screen.



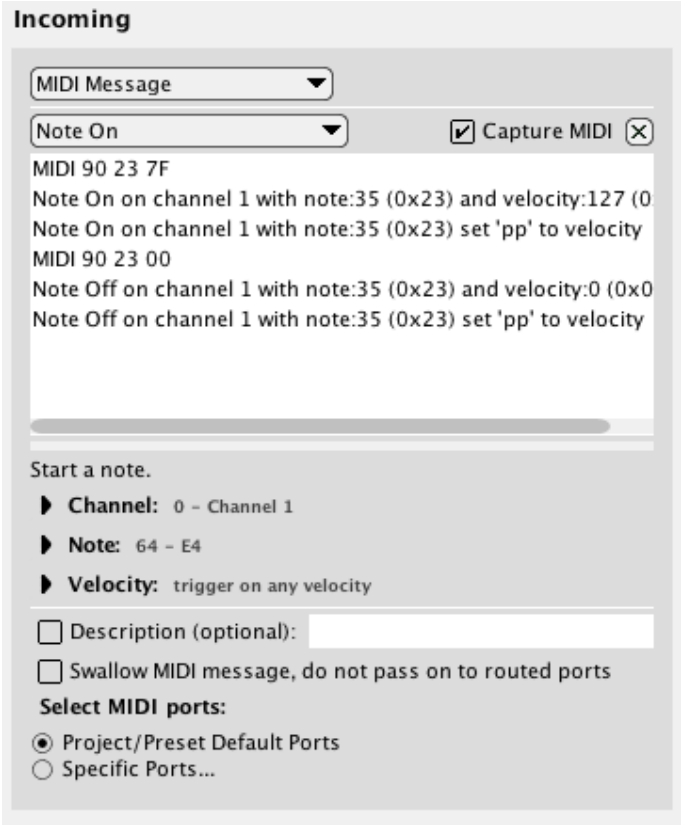
2.6 Defining Translators

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



The properties panel 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.

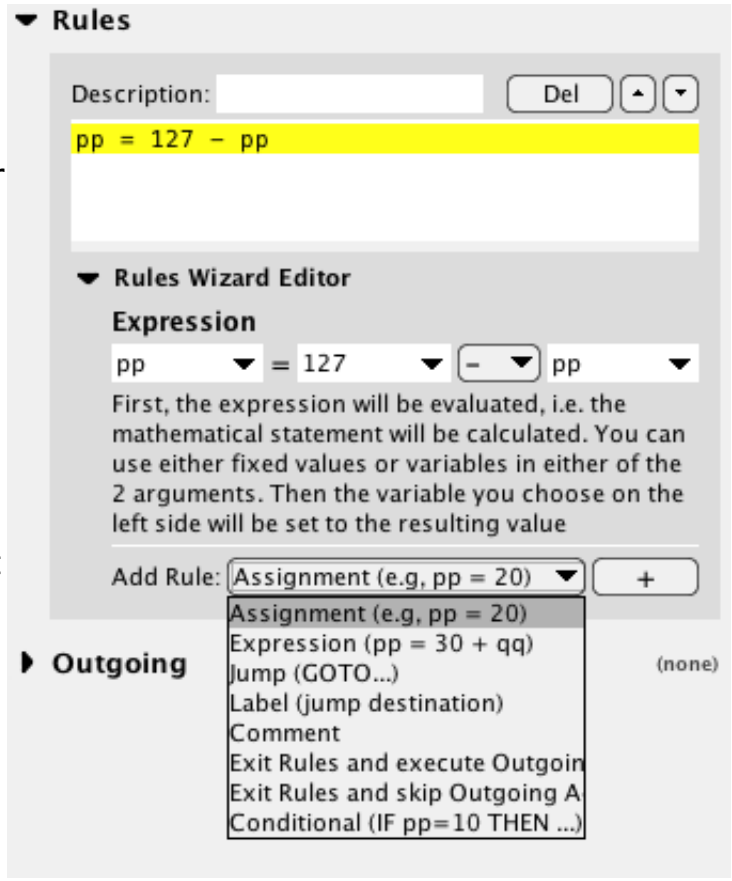
2.6.1 Incoming Action



To start, you will need to specify an incoming action to work with. Be sure the Incoming Trigger type is set to MIDI message, and 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. You may notice that every MIDI message generates multiple entries in the Capture list. That is so that you can quickly select one of several variants. For example, there can be variants with a two letter variable being used. This two letter variable can be used later in the rules section. Uncheck the Capture MIDI checkbox once you have selected the message you need.

> Incoming MIDI messages are received and transmitted in their native hexadecimal values. Variables can be assigned to any part of an incoming MIDI message trigger field.

2.6.2 Translator Rules

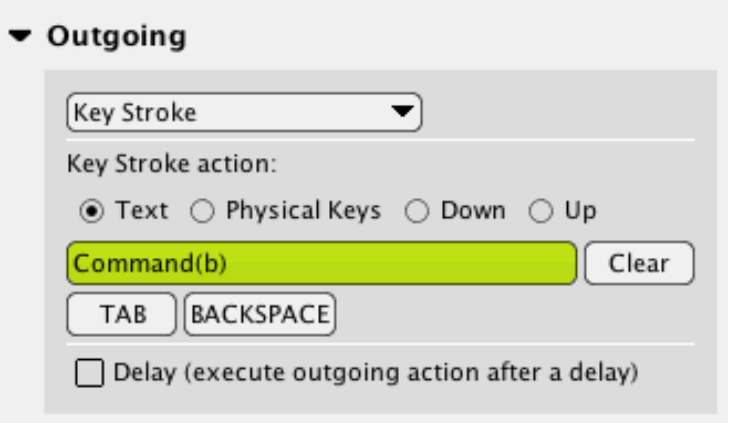


Next, select the Rules tab to view the rules entries for the translator. In this screen, you can specify rules that affect the values local to the translator, or use values stored in global variables.

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. Refer to the Translators section of the main manual to find out more about Rules.

2.6.3 Outgoing Actions



Now you need to specify what you want your outgoing action to be for the translator. Select an action type from the dropdown 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.

3 MIDI Setup Guide

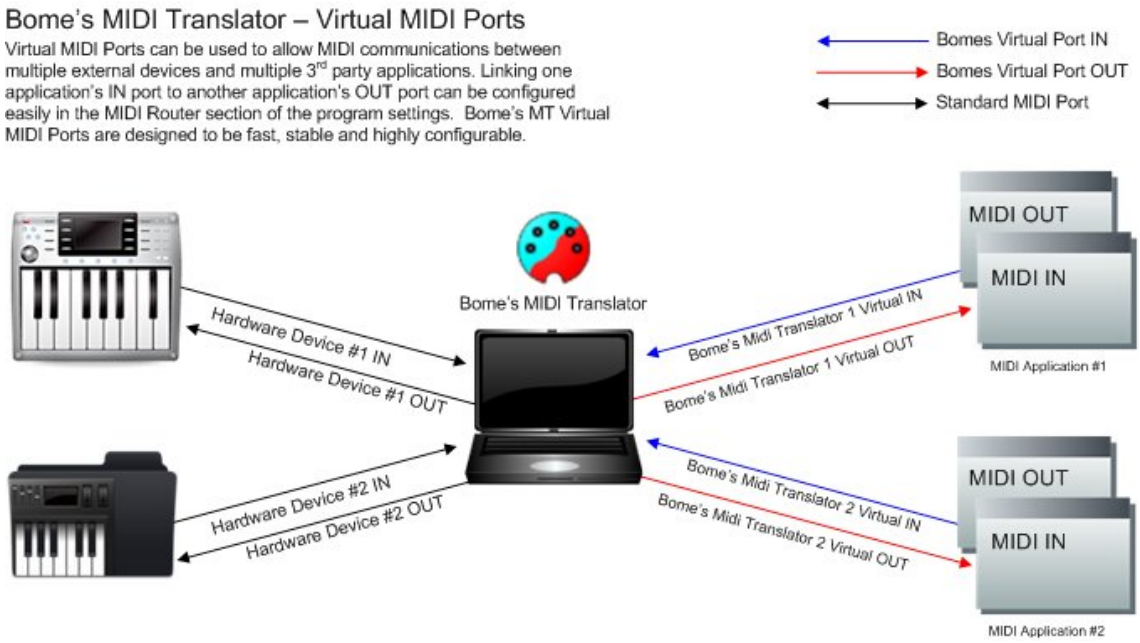
3.1 Virtual MIDI Ports

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 "loopback" 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.

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.



virtual port flow diagram

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 amongst 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 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.

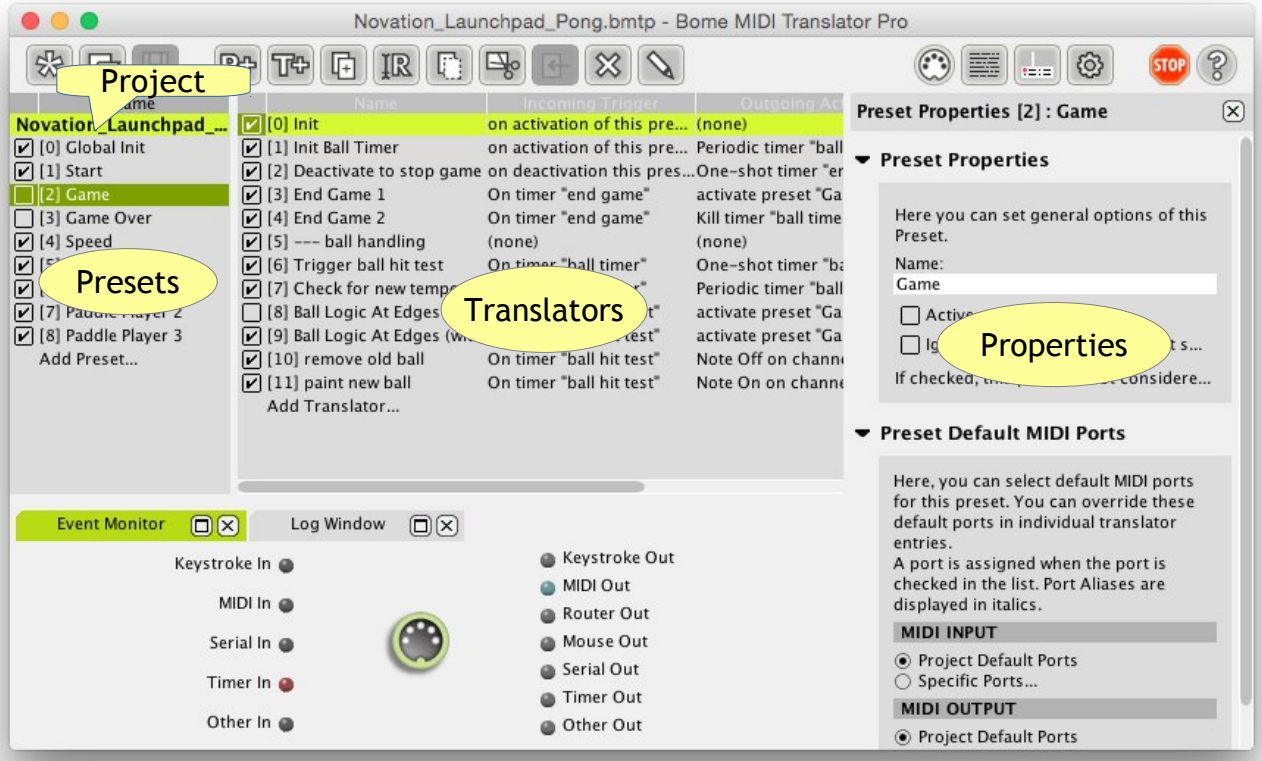
The MIDI Router is a patch panel type setup screen that allows for "patch cords" to be created between available MIDI devices and aliases. Multiple connections can be made from a single source MIDI IN device allowing MIDI data to be replicated and sent to multiple devices concurrently.

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 property editor, 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.
- **Property pane:** 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

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 of Translator or Preset (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
	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 panel at right is visible, it shows the project properties.

4.5 Preset List

Available presets are listed on the left pane, 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 a n 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 panel 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 background of the pane.

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

4.7 Properties Pane

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

To invoke the properties pane, 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  .

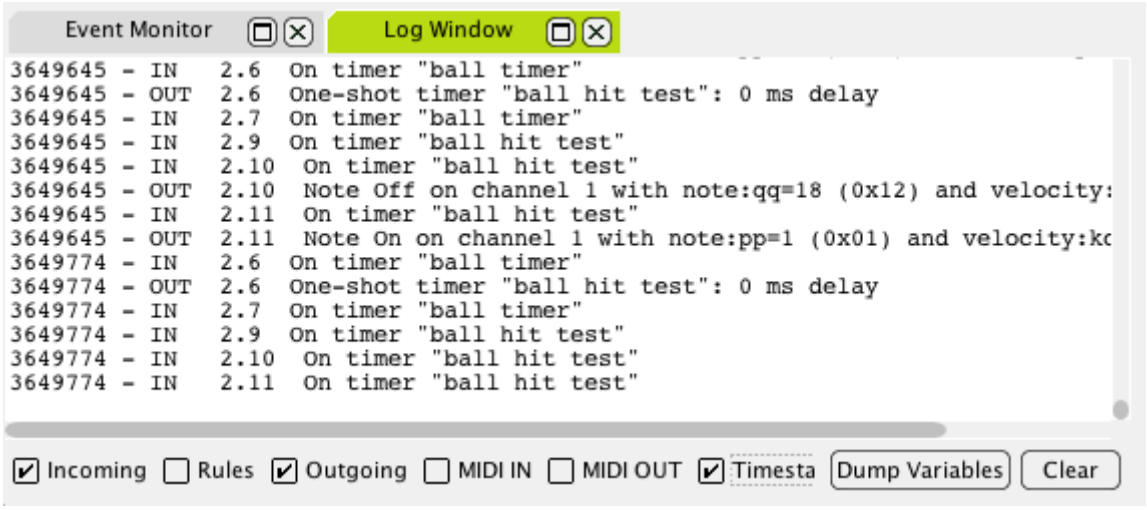
4.8 Event Monitor

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. 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.

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

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.

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 items:

- **Author 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 Action

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

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 Action 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

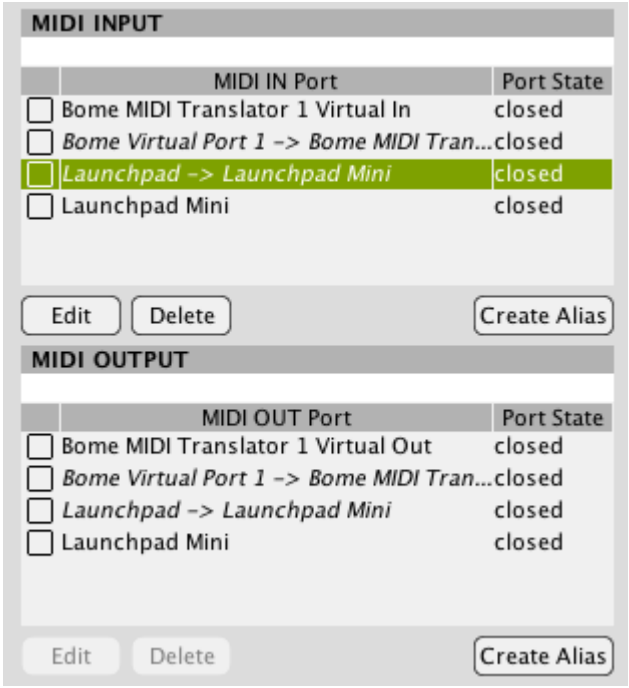
6.1 Author Info

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.

Information collected in the Author Info page includes:

- Author Name
- Author Contact Info
- Comments
  Here you can record notes about how the preset works, which variables are used for what, and possibly instructions on how to use the project.

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 normal 'Global' setup where every translator is active all the time.

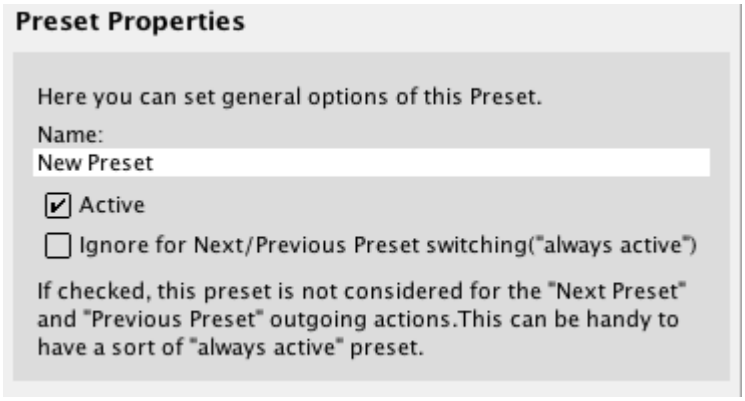
A preset's active or inactive state can be determined by looking at the checkbox next to its name. If the checkbox is checked, the preset is active and its translators are being processed. If the preset's checkbox is unchecked, it is deactivated and no processing occurs.

Presets can be selected by clicking the preset's name, and they can be activated and deactivated via the mouse by clicking the checkbox (or 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.

Presets can also be activated and deactivated via Outgoing Actions of Translators.

7.2 Always Active

Checking the *Ignore for Next/Previous Prest 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.

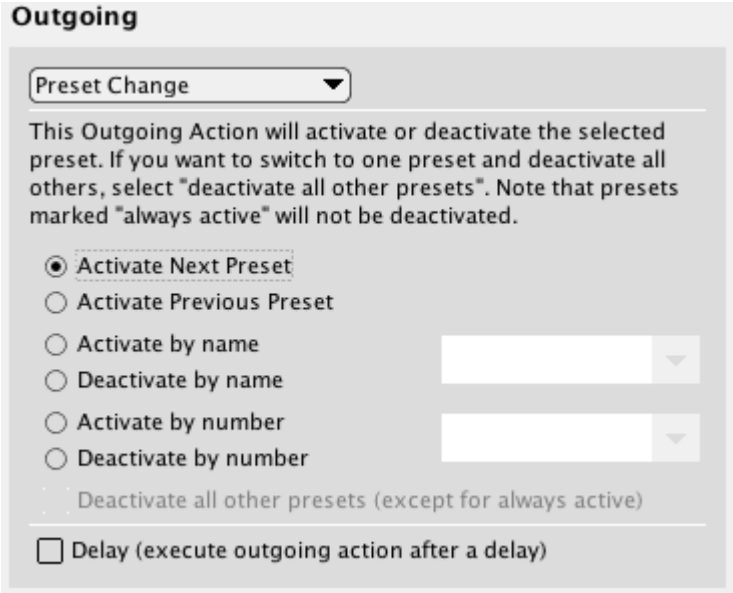


7.3 Changing Presets

Presets can be activated and deactivated by clicking the checkbox next to the preset name. This is the most direct way of working with Presets. However, mouse and GUI access is at many times at a premium, so Bome MIDI Translator features the capability to switch and work with Presets via the 'Preset Change' outgoing Translator action.

Presets can be activated via Preset Change action by one of three primary methods:

- **Activate Previous/Next 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.
- **Activate/Deactivate 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.
- **Activate/Deactivate By Number**
  Presets can be enabled or disabled by number, which can be specified by a unique local or global variable (see the rules section for more information). This outgoing preset change action is useful for more complex MIDI Translator scripts 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.



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

7.4 Default MIDI Ports

Individual Presets can have 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 Output ports are useful if you are working with multiple software programs or outboard devices, and you wish to divide and manage translator data amongst them.

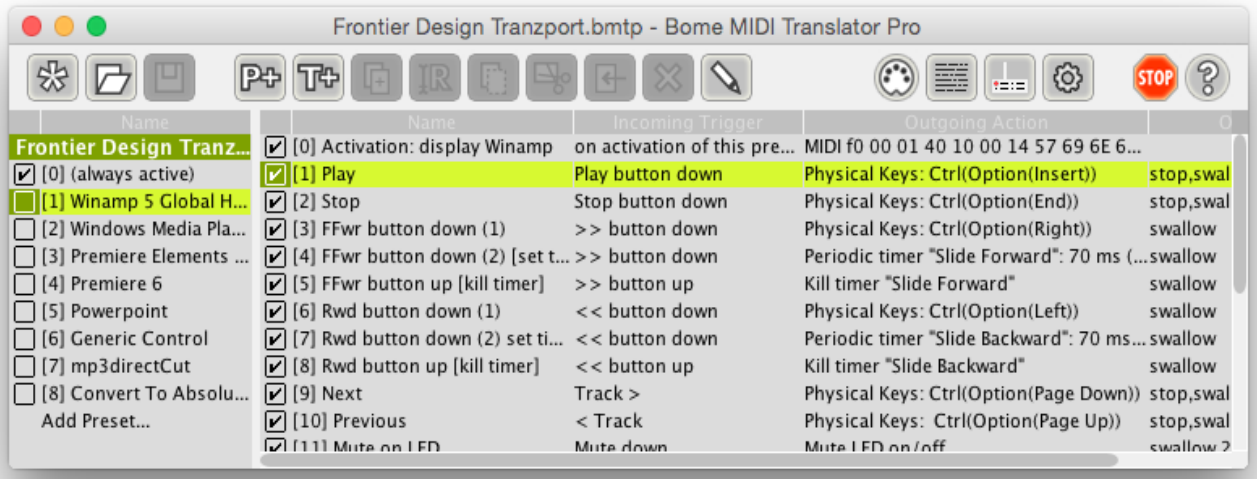
Defining Default Ports for a preset is simple. First, select the Preset you wish to change, then check the properties for the Default MIDI Ports sections. All the ports that you check will be defined as preset default ports.

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.

8 The Translator Entry

8.1 Overview

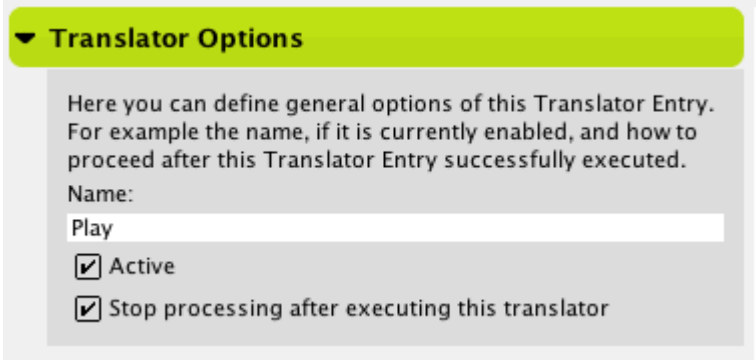
Translator Entries comprise the core functionality of Bome MIDI Translator. In simple terms, translators listen for an 'incoming action', optionally do some processing on the incoming action (see the Rules section of the manual), and then optionally output an 'outgoing action.' Translators are limited to a single incoming action and outgoing action, but you can easily create multiple translators 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.



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 and activation rules. The Translator name is not a unique value, 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 13.6.2 Use "Stop Processing").

8.3 Incoming Actions

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 the next chapter for using the individual action types.

8.4 Rules

This is 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 chapter 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 the next chapter 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.

Rules

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

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.

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.

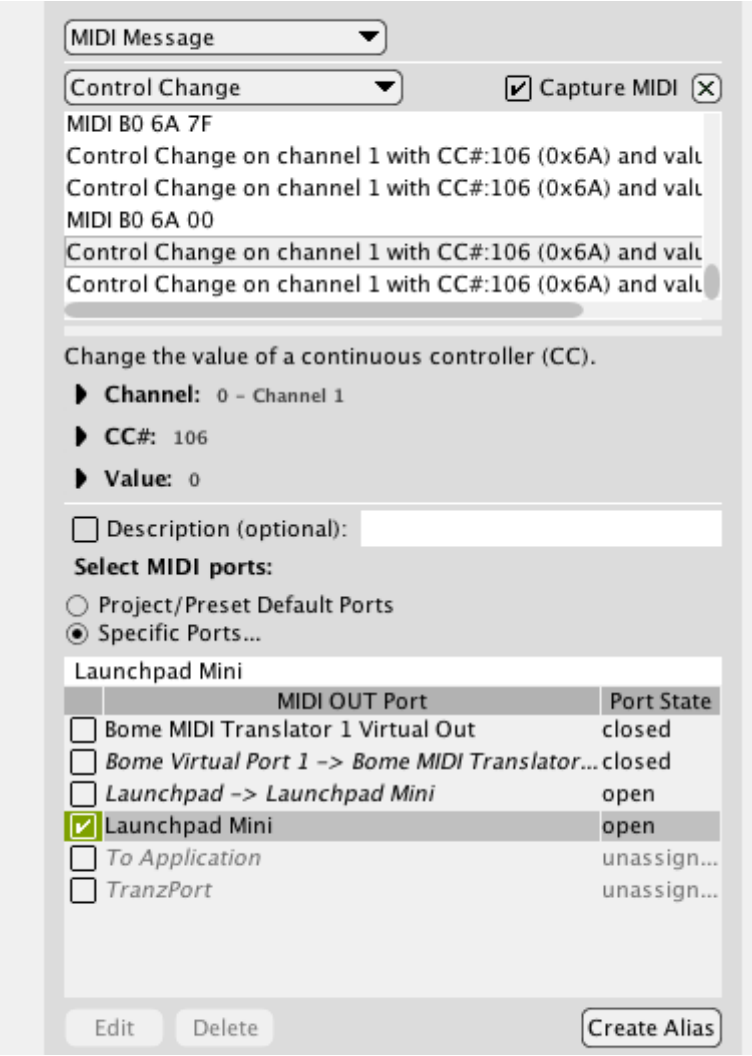
It is good practice to add comments to longer Rules sections so that you'll remember what the rules actually do. A comment line starts with //.

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

9 Actions

9.1 MIDI

9.1.1 Incoming MIDI



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. Standard MIDI messages will be listed three times:

1) the raw MIDI message (hexadecimal notation)

2) simple mode MIDI message

3) simple mode MIDI message with value set to a variable

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

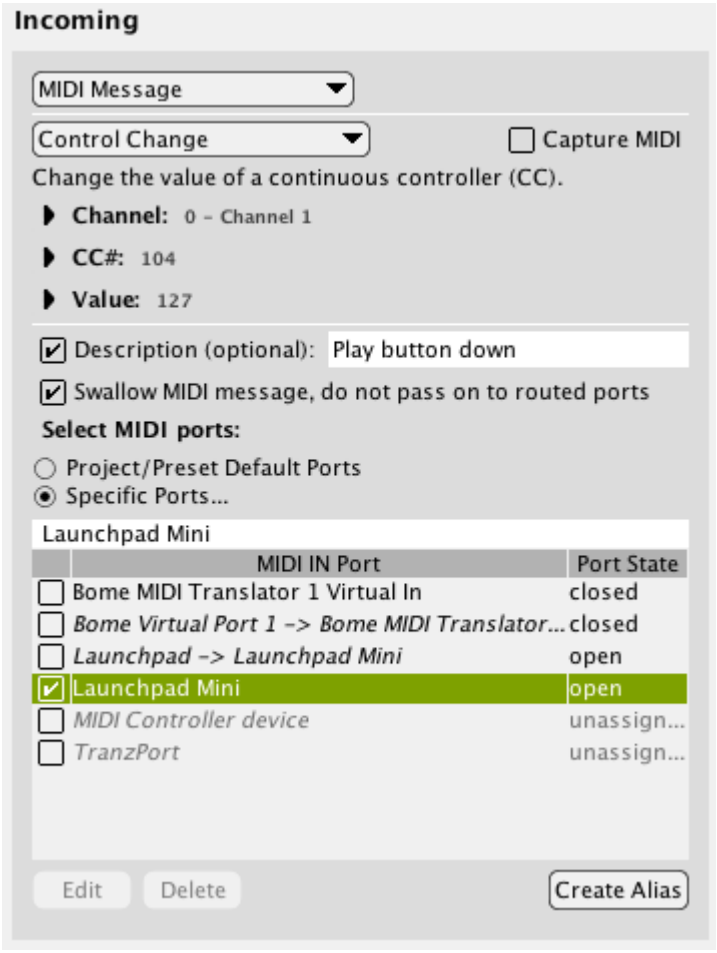
Uncheck "Capture MIDI" (or press Alt+C) to stop displaying incoming MIDI data. Click the X button to close the Capture MIDI panel.

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

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 variables beginning with the letters 'g' through 'n', plus 'y' and '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

This option indicated that you wish to NOT retransmit the incoming MIDI message on to the MIDI Thru destinations specified in the Project Routing connection screen. So, when an incoming MIDI message matches at least one Incoming MIDI trigger, the message will not be sent to the MIDI Router. By specifying None as outgoing action, you can filter out MIDI messages from the router.

Select MIDI Ports

The Incoming MIDI trigger 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 to use Preset Default MIDI ports where possible.

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 to use 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.

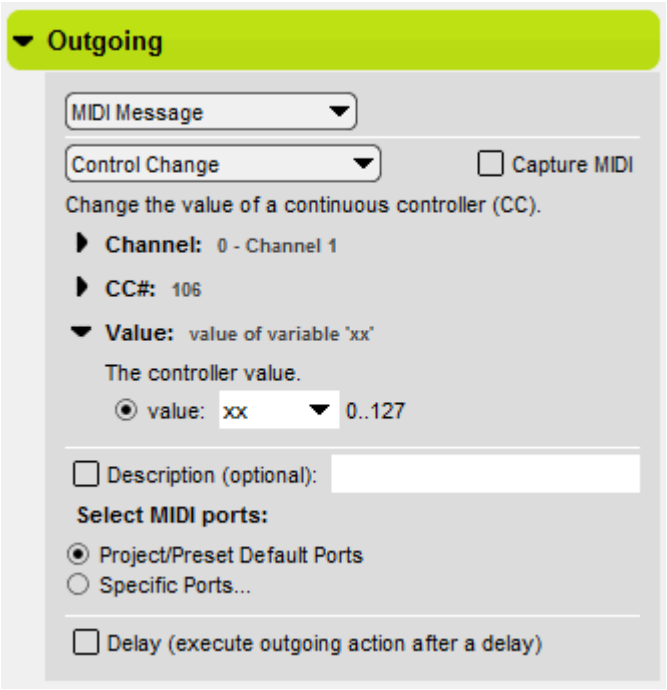
ACTION STRING	TYPE	CHANNEL	CONTROLLER/ NOTE/PROGRAM	VALUE
9F 6F pp	Note On	16	111	ANY
BA ww 7F	Controller	11	ANY	127
B4 xx pp	Controller	5	ANY	ANY
C4 nn	Program Change	5	ANY	n/a
pp qq rr	any 3-byte MIDI message	any	any	any

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

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.

Raw MIDI / System Exclusive

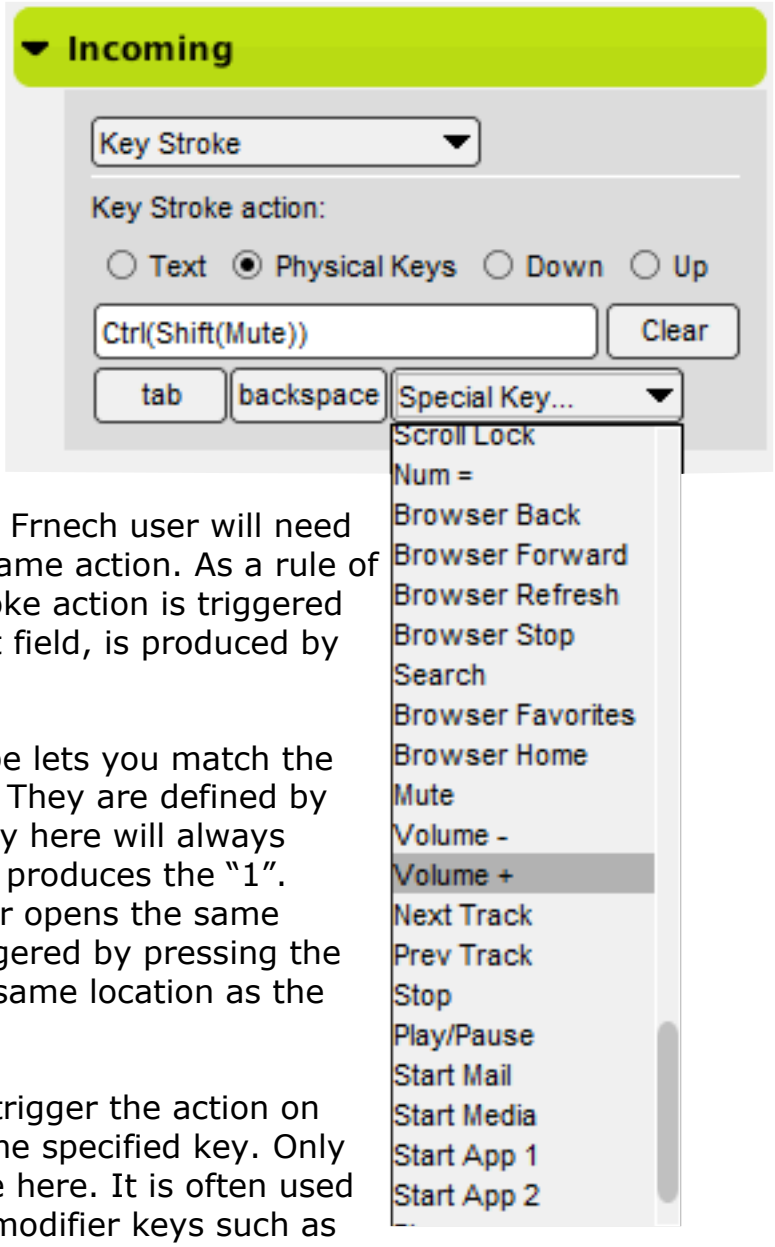
Similar to the Incoming MIDI trigger, you can specify the outgoing MIDI message as a raw MIDI message in form of a string of hexadecimal numbers. Check the MIDI Specification or online sources for more information on raw MIDI message definitions.

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 Keystrokes

9.2.1 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.

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, Frnech user 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.

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.

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.

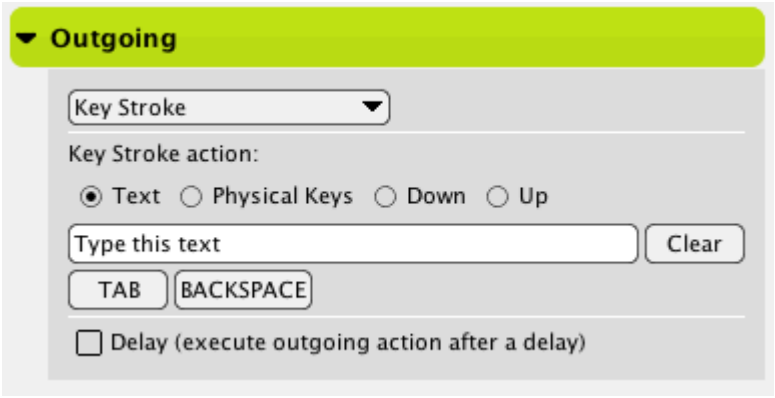
For the Text and Physical Keys types, you can specify arbitrary sequences and key combinations to serve as a trigger. Parenthesis are used to mark simultaneous presses of keys. This can be conveniently used to trigger on **keyboard shortcuts**. For example, "Shift(Ctrl(A))" will only trigger if you press 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.

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** dropdown list allows you to enter a special key which might not be present on your keyboard.

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

9.2.2 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** dropdown 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.

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.

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”).

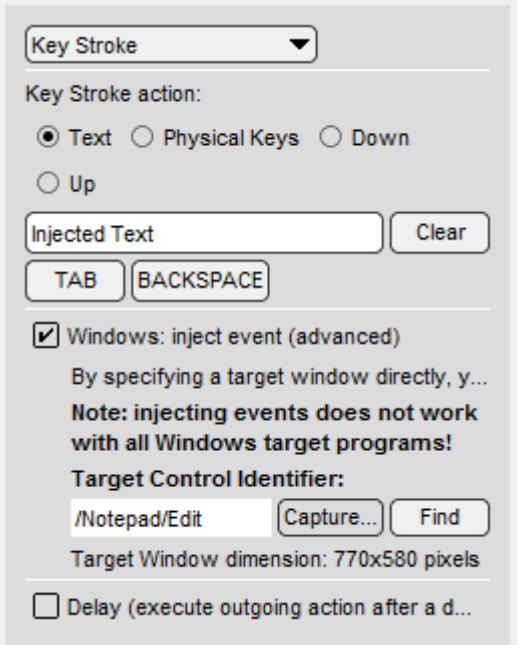
Now switch to a text editor and press the key on your MIDI Keyboard down. As long as you keep it pressed, the letter "X" will be typed repeatedly. Once you release the key, the keystroke emulation stops, too.

9.2.3 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 to 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, SPACE to a button sub window, effectively clicking the button.

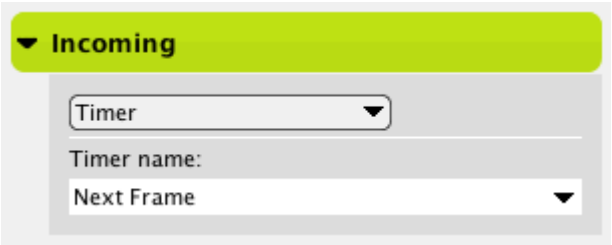
9.3 Timer

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

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

1. Wait for the *Initial Delay*.
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 and the timer's name will be executed.
4. If this is a *Once* timer, stop timer.
5. If this is a *Multiple Times* timer and number of executions has reached the *Repeat Count*, stop timer.
6. Wait for the *Repeat Delay*.
7. Start over at 2.

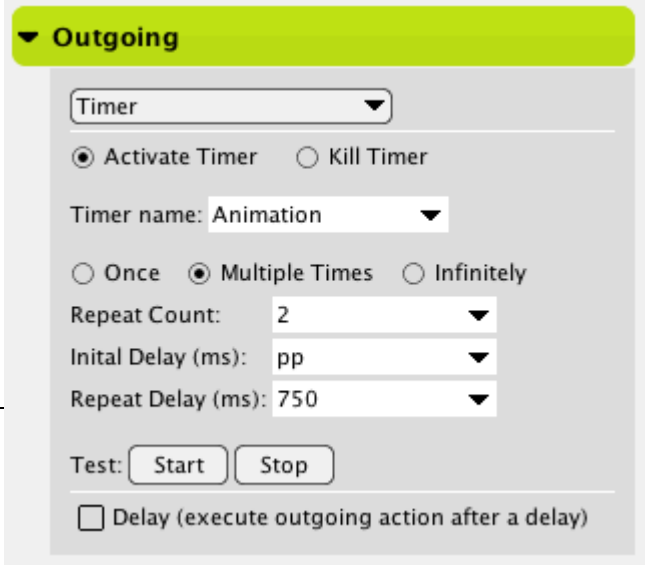
9.3.1 Incoming Timer



Incoming Timer actions are events that will trigger once or multiple times automatically depending on the Timer settings. 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 only option is the selection of which Timer to use. Most of the Timer options are defined in the Outgoing Timer Action screen, which is covered below.

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

9.3.2 Outgoing Timer



There are two main types of 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 wanted to repeat the 'Up' arrow key as long as a condition is met, you would first create an outgoing Timer action that set the repeat rate, then you would create a new translator which would output the keyboard emulation for the 'Up' arrow key - using your existing Timer translator as the incoming action.

Timer parameters include repeat occurrence (once, multiple times, indefinitely), initial delay, repetition delay and testing functionality.

If you start a Timer that is already running, the delay time will be refreshed with the new delay time.

One-shot 0ms Timer

A trick is to use a one-shot timer with 0 delay: 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. For example, if you have a series of outgoing actions to be executed from multiple triggers, define the same 0ms timer as Outgoing Action for all trigger Translators, and use that timer as Incoming Action in all Translators of the series of outgoing actions. See also chapters 13.4 Timer with 0ms and 13.5 Multiple Actions in one Translator.

Timer Initial Delay vs. Outgoing Action delay

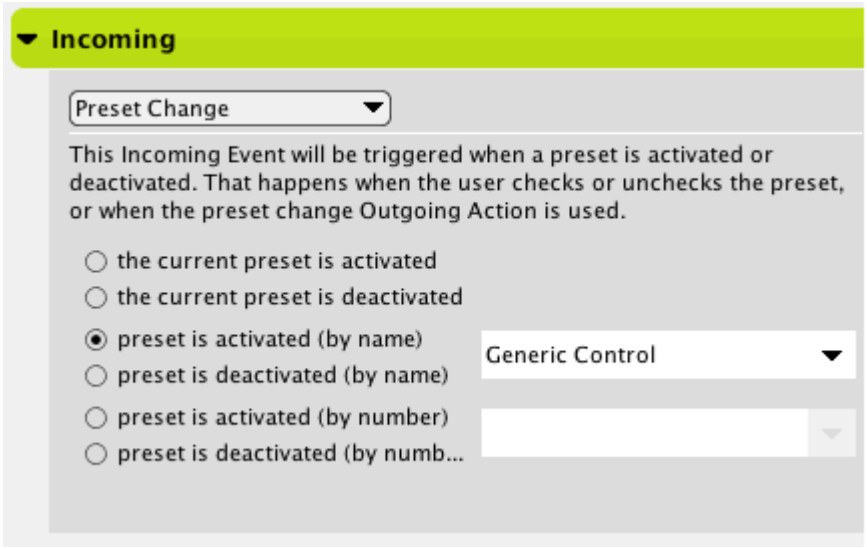
Note that for starting a timer, you can specify an initial delay, and there is also the possibility to specify an action delay (see Delaying Outgoing Actions). Those two delays add up, but there is a big difference: by delaying the outgoing action, the timer will not exist before the outgoing action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

9.4 Preset Change

9.4.1 Incoming Preset Change

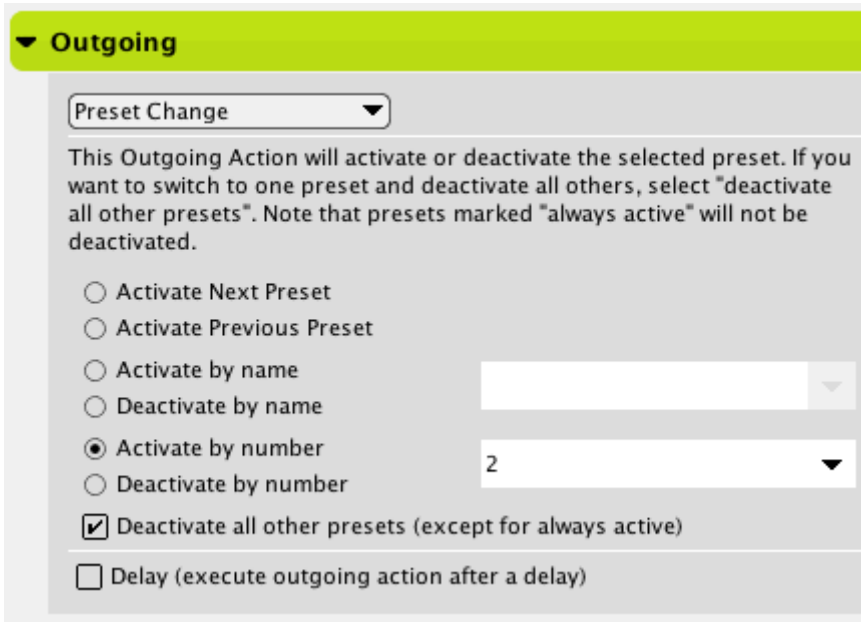
Preset change incoming actions are actions that are activated when the preset is changed through some means. Preset Change Actions are useful for "one-off" type of actions that only occur once at the very beginning of the preset change. These actions are often 'reset' actions that could either redefine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

When loading or restarting a Project, a preset change event is triggered for all initially active presets.



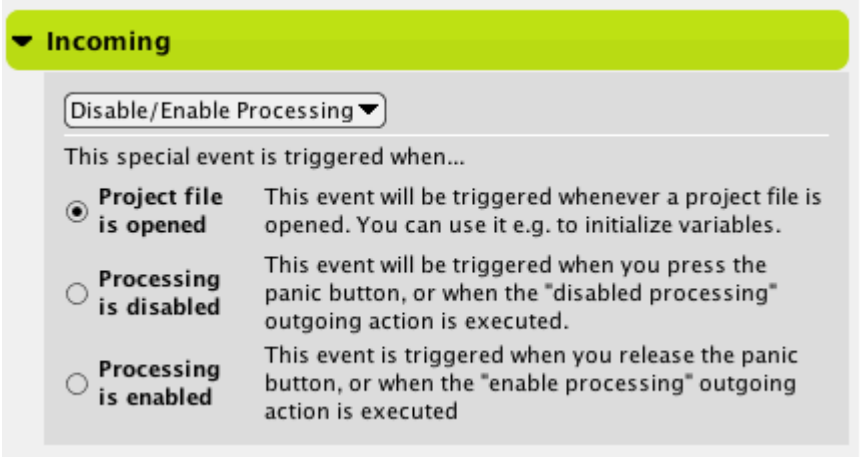
9.4.2 Outgoing Preset Change

Preset change outgoing actions have the capability of managing project presets, either by activating certain ones, deactivating certain ones, 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.



9.5 Disable/Enable Processing Actions

9.5.1 Incoming Disable/Enable Processing



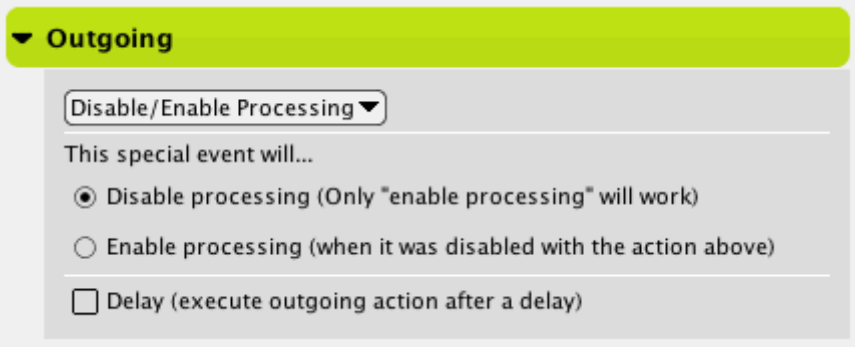
This incoming action is triggered in response to program events. 'Startup' actions occur when the project file is loaded. Certain actions that need to take place first before any other actions, and only once are best defined by here. For example, it's a good idea to initialize variables to start-up values when the project file is opened.

The “processing is disabled” type is triggered when you press the panic/STOP button, or the “disable processing” outgoing action was executed. You can use it to send additional STOP messages or to do other housekeeping.

Consequently, the “processing is enabled” type is triggered when processing restarts by either pressing the STOP button again, or when the outgoing action “enable processing” is executed.

Outgoing Disable/Enable Processing

There are two functions - disabling MIDI Translator Processing and enabling MIDI Translator Processing. The program as a whole can be bypassed using the 'Disable Processing' outgoing action, preventing any of the translators from activating. If MIDI Translator is disabled in this way, the only way to re-enable it is by using the 'Enable Processing' outgoing action defined in a different translator.



9.6 Mouse (Outgoing)

Included in Mouse-type outgoing actions are multiple types of actions: **Movement, absolute positioning, button clicks and wheel**.

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

9.6.1 Movement

Movement events can be transmitted to the system mouse pointer using this mouse event type. 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.

You can also use variables for movement.

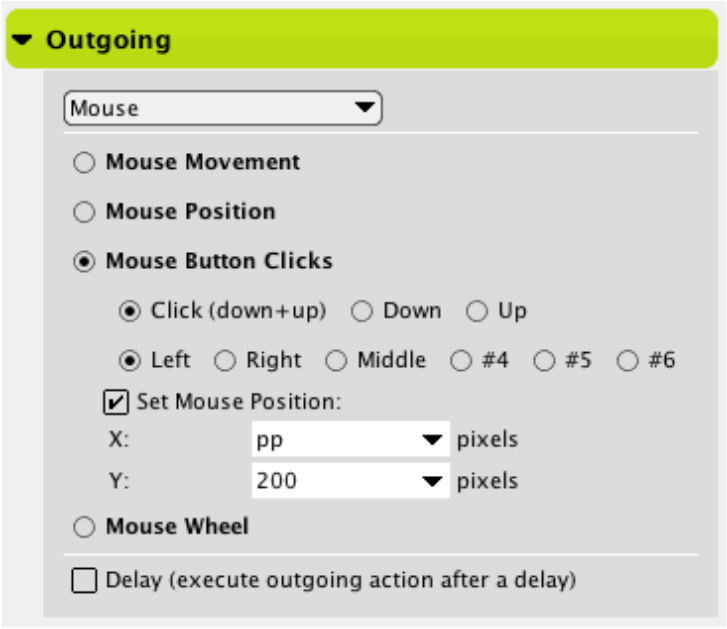
9.6.2 Absolute Position

The absolute positioning type allows for the mouse to be moved to a pre-determined 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 1024x768 screen, the exact center of the screen would be absolute position 512x384).

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

On Windows, you can use the Capture button, then move the mouse to enter a specific mouse position. Click the mouse to retain the current position in the outgoing action.

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



9.6.3 Button Clicks

Button clicks can be emulated using this mouse outgoing action. 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 "Mouse Position" and effectively executes two actions at once.

9.6.4 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.

9.6.5 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.

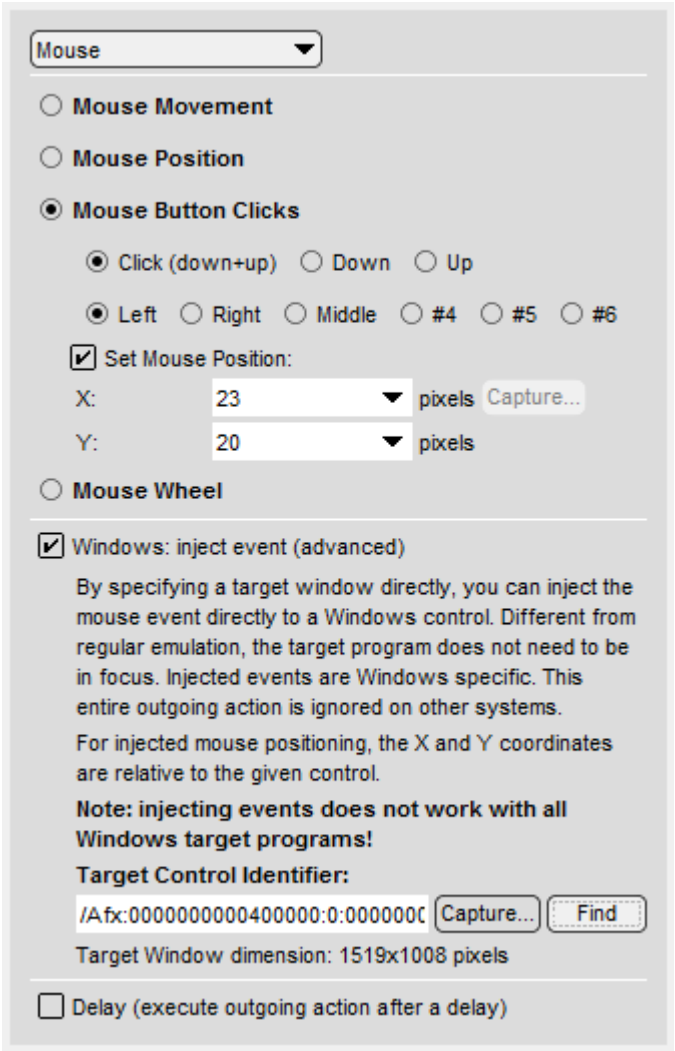
Injecting mouse events can make your projects much more robust because it will still work when (accidentally) another program is activated, and you can control multiple programs with mouse events without the need to switch the active window.

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 to 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.

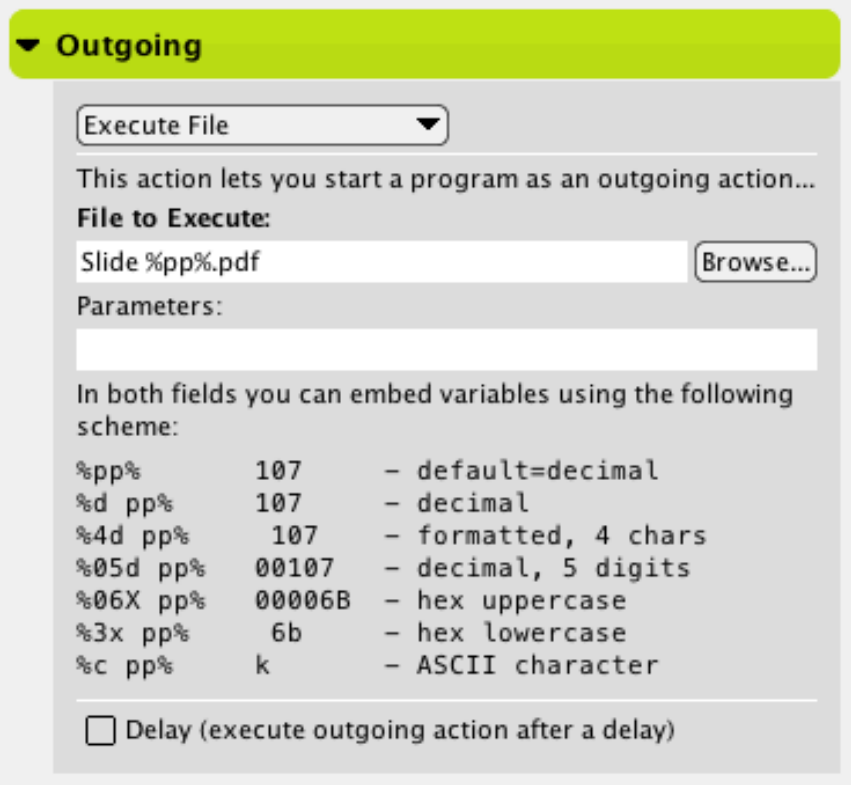
Sometimes, it works to directly inject, say, a mouse button click event to a button sub window, effectively clicking the button.



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

9.7 Execute File (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 (usually surrounded by quotation marks).



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 to 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.

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 number 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.

```
Filename:          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.).

9.8 Serial Port

With the Serial Port trigger and action, 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.

9.8.1 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
```

9.8.2 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 to use 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.

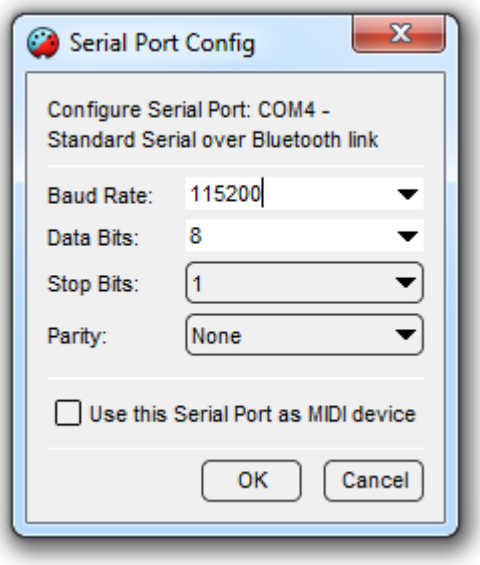
To define the physical port to an alias, click the *Edit* button. In order to remove an alias, use the *Delete* button. If the alias is used in other translators, it will likely be recreated. Also, if you load a project file with the deleted alias, it will be recreated.

9.8.3 Configuring the Serial Port

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

Serial port configuration is per serial port, not per alias.

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



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.

9.8.4 Using a Serial Port as a MIDI Device

There are devices which send MIDI data over a standard serial port. For those scenarios, you 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.

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

9.8.5 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.

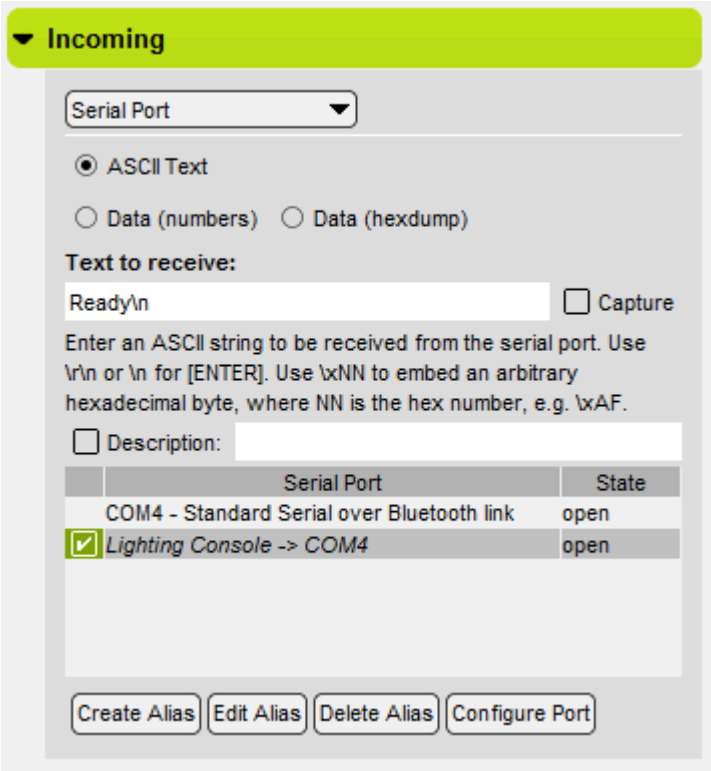
> The Capture function always displays received serial port data, not the data which is sent out by an outgoing action.

9.8.6 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).

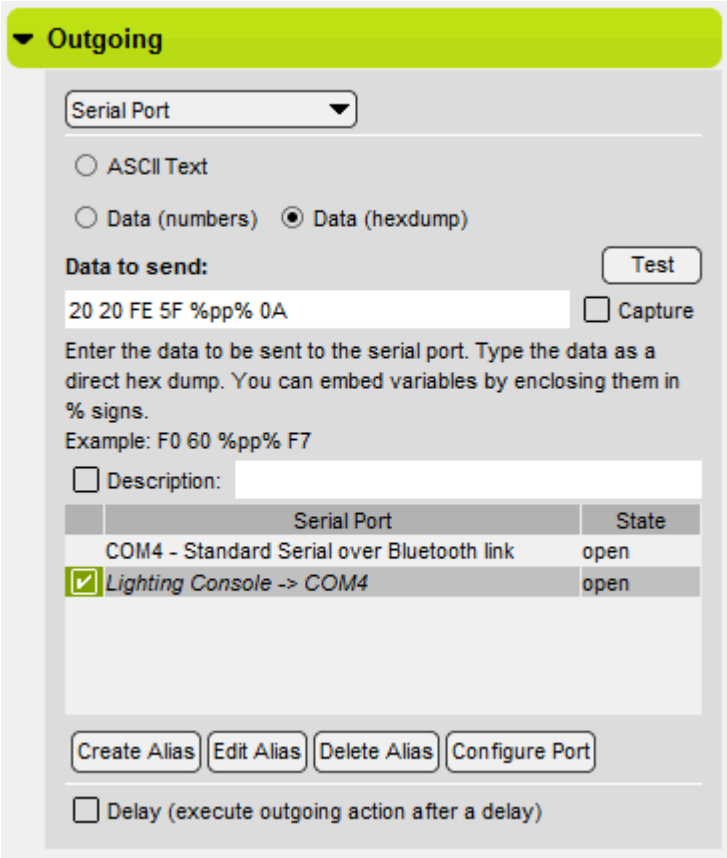
Now, in the rules and outgoing action, you can use and modify these variables, or, for global variables, use them in different translators.

9.8.7 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:

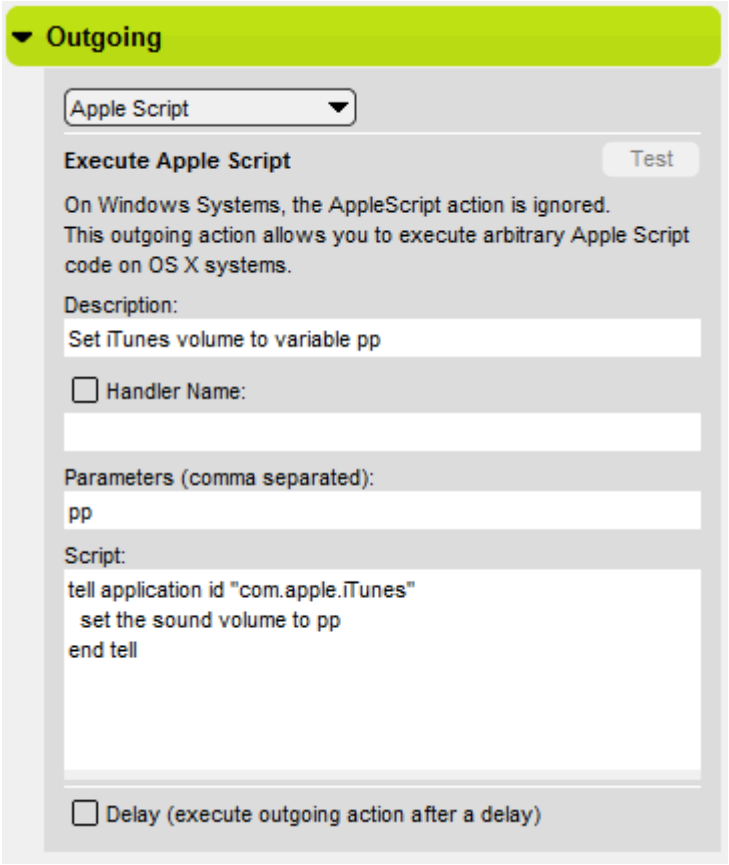
```
decimal:          10 20 30 56 67 255 126 0
hexadecimal:      0A 14 1E 38 43 FF 7E 00
```

9.9 AppleScript

9.9.1 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 OS X.



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

You can also define global AppleScript handlers in the project properties. These handlers are available in all AppleScript actions (see next chapter).

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 will not be reflected back to the MIDI Translator engine.

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.

Call Outgoing AppleScript from other Outgoing Actions

The AppleScript outgoing action is internally implemented as a handler. You can optionally give the handler a name so that it is available from other AppleScript actions and from the global AppleScript section.

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 it). if you need to self-reference MIDI Translator application, use that identifier rather than hard coding the name.

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).

9.9.2 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>
```

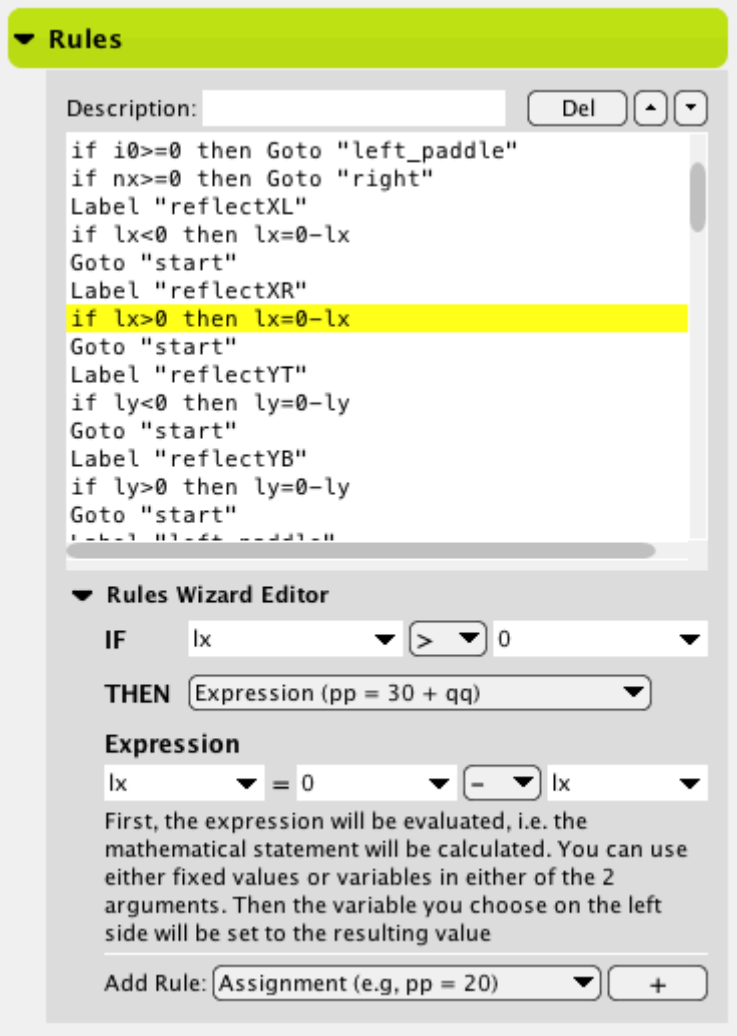
You can only manipulate global variables.

> For external AppleScript support to work, you must have the application 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 "Bome MIDI Translator Pro"
        set variable "g0" to 123
        set asVar to get variable "g0"
end tell
log asVar
```

10 Rules and Variables



translator rules

10.1 Overview

Translators are comprised of three main sections: incoming actions, outgoing actions, and rules. This part of the documentation covers Rules, how to use and to be related 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 Labels and Jumps 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 (Jump 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.

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.

10.2.2 Jump

Redirects the processing of the rules to a 'Label' point.

A Label name may be typed in directly, or an existing label jump destination may be picked from the drop-down box.

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

10.2.3 Label

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

10.2.4 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.

10.2.5 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.

10.2.6 Conditional

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
- Jump
- 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)

10.3 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

Local Variables are defined by character combinations in the following range:
oo-xx (*example: pp, ss, ww, etc...*).

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 simulatenous 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, ..., na...n9, and ya-y9, za-z9

Variable names where both letters are the same are local variables.

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

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 way 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.

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

- Startup
- Appearance
- 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 icon:**
  If enabled, a small MIDI Translator icon is visible in the tray (Windows) or menubar (OS X). You can use it to quickly open MT's window, or to use it instead of the main icon in task bar / dock (see previous option).
- **Minimize to tray/menu bar:**
  If checked, and the tray/menu bar icon is visible (see previous option), minimizing the program will "hide" it in the tray (Windows) or menu bar (OS X). The main program icon in the task bar (Windows) or dock (OS X) is hidden then. Double-click the tray/menubar 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 MT Pro had been minimized to – i.e. the tray / menu bar icon.
- **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.

11.3 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.

11.4 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

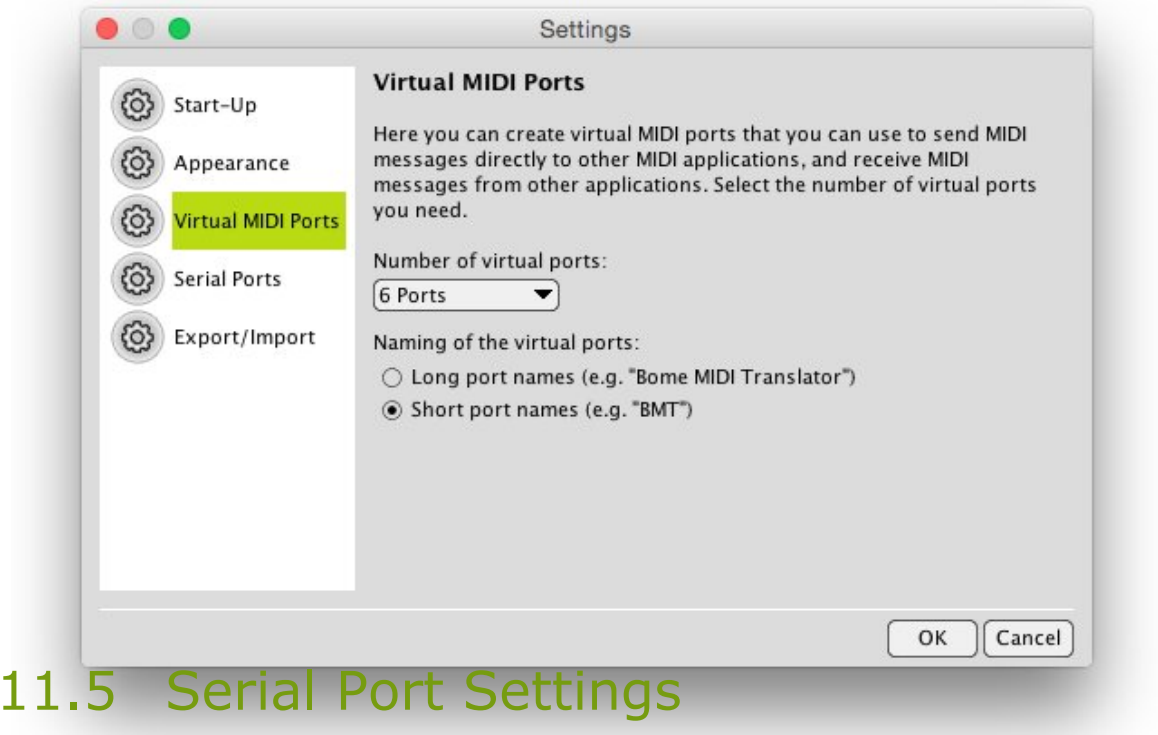
In the drop-down list, simply choose how many pairs of virtual MIDI ports you require in your processing (1 virtual MIDI port = 1 MIDI IN, 1 MIDI OUT), select Apply and follow the on-screen hardware installation instructions similar to the product installation covered in the Quickstart guide.

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.



11.5 Serial Port Settings

In the Serial Port settings, you can configure the serial ports. The port setting is stored in the project file.

11.6 Export/Import Settings

11.6.1 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.

11.6.2 Create .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.

11.6.3 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.

11.6.4 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 OS X, 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.

11.6.5 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 app file is “MIDI Translator.exe”, rename the settings file to “MIDITranslator.bmts”. and copy it side-by side to the same folder. Note that Windows will usually not show the .exe extension.

On OS X, 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.

11.7 Reset

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

11.7.1 Reset All

Clicking this button will reset all settings to their defaults.

11.7.2 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.

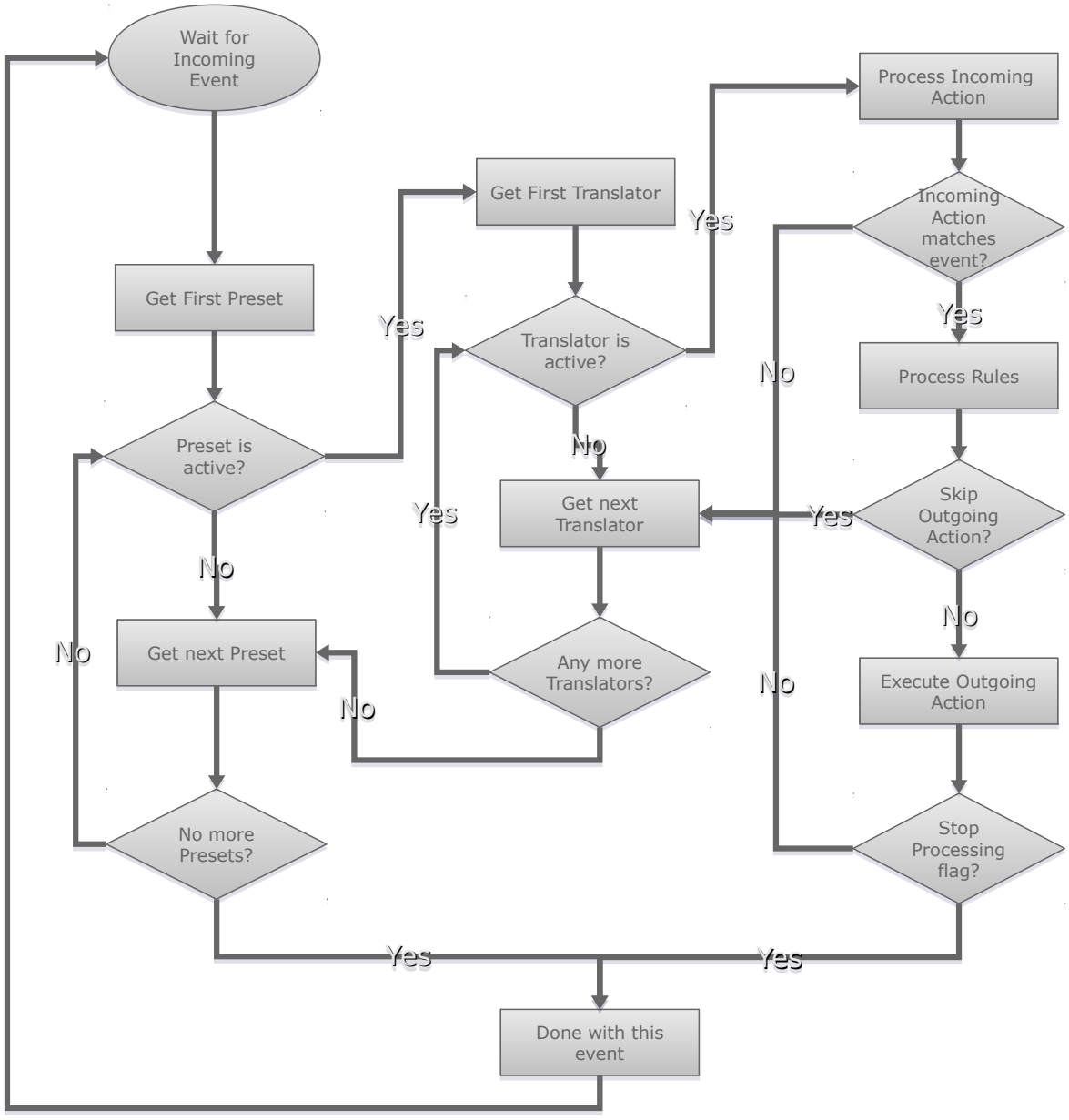
Processing the Incoming Action is mainly a check to see if the event matches the definitions in the Incoming Action, e.g. to compare the MIDI message in the Action with the MIDI message of the event. Some Incoming Actions may also set local or global variables (like the MIDI Incoming Action).

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.

Flowchart Incoming Event Processing



Template by Marc Carson, www.marccarson.com.

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. OS X 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 OS X)
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
   OS X: 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 OS X) 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

If you need virtual MIDI ports, we recommend to 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. For that, copy all contents of the installation folder (e.g. C:\Program Files\Bome MIDI Translator Pro) to the thumb drive.

13.3.2 OS X

Installing MIDI Translator on an external disk or a USB thumb drive is not a big problem for the OS X 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 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.

13.3.6 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 delay. It will be processed very efficiently, usually already in parallel to the current event (see also Parallel Processing).

That can come in handy for a quick way to trigger a series of actions with one or more incoming actions:

```
Translator 0:
         Incoming: MIDI message 1
         Outgoing: Timer "Execute Series", 0ms delay
Translator 1:
         Incoming: MIDI message 2
         Outgoing: Timer "Execute Series", 0ms delay
Translator 2:
         Incoming: Timer "Execute Series"
         Outgoing: out 1
Translator 3:
         Incoming: Timer "Execute Series"
         Outgoing: out 2
Translator 4:
         Incoming: Timer "Execute Series"
         Outgoing: out 3
```

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

13.5 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 Timer with 0ms.

13.6 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.

13.6.1 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.

13.6.2 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.

13.6.3 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.

13.6.4 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.

13.6.5 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.

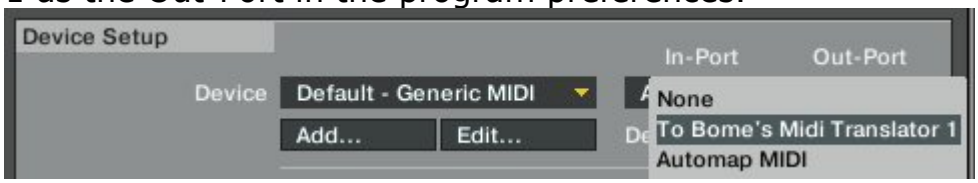
Turn off the Log Window when actually using MIDI Translator in action, to ensure best performance.

14 Usage Examples

14.1 Traktor / Ableton Live Sync

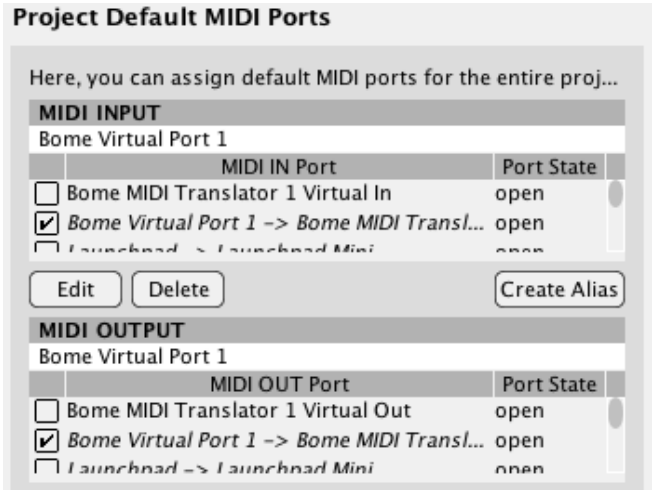
Virtual MIDI ports are available for use as unidirectional MIDI ports, so you only need one virtual MIDI port to get MIDI data into a target application like Traktor, and to get data back from it.

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.

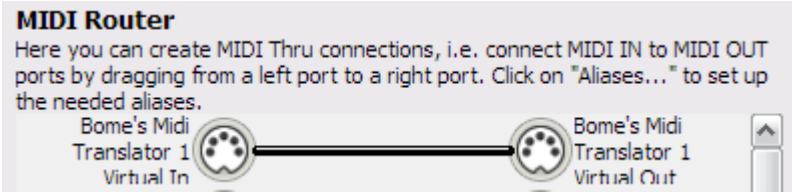


   Make sure you have the program set to transmit the clock signal by going into the MIDI Clock category of the preferences dialog and making sure "Send MIDI Clock" is checked. Also make sure that you have MIDI clock playing by accessing the metronome section of the user interface and clicking the "Play"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

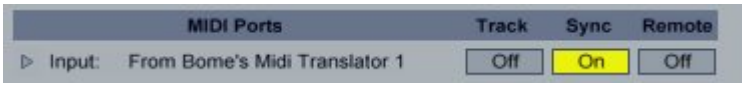
2. The next step is to configure Bome MIDI Translator to activate the virtual ports we wish to use coming IN, going OUT and then to link them via the program MIDI Router. Activate the Virtual MIDI port OUT and IN devices by going into Settings / MIDI Ports and putting a checkmark next to each MIDI device.



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 Reference

15.1 Terminology

- **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.
- **Preset**: A Preset is a collection of Translators. There are individually named Presets, which may be switched and activated separately, as well as an '[always active]' preset, which is perpetually active.
- **Translator**: A Translator is an individual "rule" defining the translation of a single 'Incoming Action' event through to a single 'Outgoing Action' event.
- **Name:** The Name refers to the unique name given to an individual Translator or Preset.
- **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 expired, Preset Changes, ...
- **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.
- **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.

15.2 Keyboard Shortcuts

On OS X, the shortcuts are used with the Command key instead of the Ctrl key.

Global	
Shift+Esc	MIDI Panic - Stops all MIDI information immediately
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
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
Lists	
Ctrl+C	Copy to clipboard
Ctrl+V	Paste from clipboard
Ctrl+X	Cut to clipboard
Ctrl+D	Duplicate selected preset/translator
Del/Backspace	Delete currently selected object
F2	Rename currently selected object
Ctrl+A	Select All objects in active viewing area
Ctrl+Up/Ctrl+Down	move selected object up/down in list view

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

15.3 Command Line Switches

/debug /info /error /timedebug	debugging output
/silent	no output
/nodebug /nosilent	reverse the meaning
/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.
/project <filename>	load .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.
/midiin <MIDI dev>	select the named MIDI device on startup for input. You can use additional /midiin commands to open multiple ports.
/midiout <MIDI dev>	select the named MIDI device on startup for output. You can use additional /midiout commands to open multiple ports.
/bmidi <num ports>	select the number of virtual MIDI ports you wish MIDI Translator to use (0...5). 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 do not remove ports.
/noShow	do not activate and show main window if it is currently hidden or minimized
/close	carry out the command line paramaters and then exit (do not start the MIDI Translator window).

15.4 Menu Reference

- **File**
  - **New:** close the current project and start a fresh one
  - **Open:** Open Project - Opens an existing MIDI Translator project file
  - **Save:** Save Project - Saves current project, prompting for a name if first save
  - **Save As:** Save Project As... - Saves current project as a new 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:** Close Project - Closes current project, prompting for save if applicable
  - **Restart Project:** initializes the loaded project as if you'd just loaded it: clear global variables, and invoke the "project started" event.
  - **Project Properties:** Show the Project Properties in the right properties pane
  - **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:** Submenu - 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** properties pane 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
  - **Close all MIDI Ports:** disable all currently used MIDI ports
  - **Open Used Ports:** 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.
  - **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 right pane
  - **Preset Properties:** show the preset properties in the right pane for the currently selected preset

- **Translator sub-menu:** show Translator properties in the right pane and focus on General options, Incoming action, Rules, or Outgoing action.
- **Settings:** open the settings window with general options for the entire program. On OS X, 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)
  - **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 OS X, the About screen is available from the left application menu.