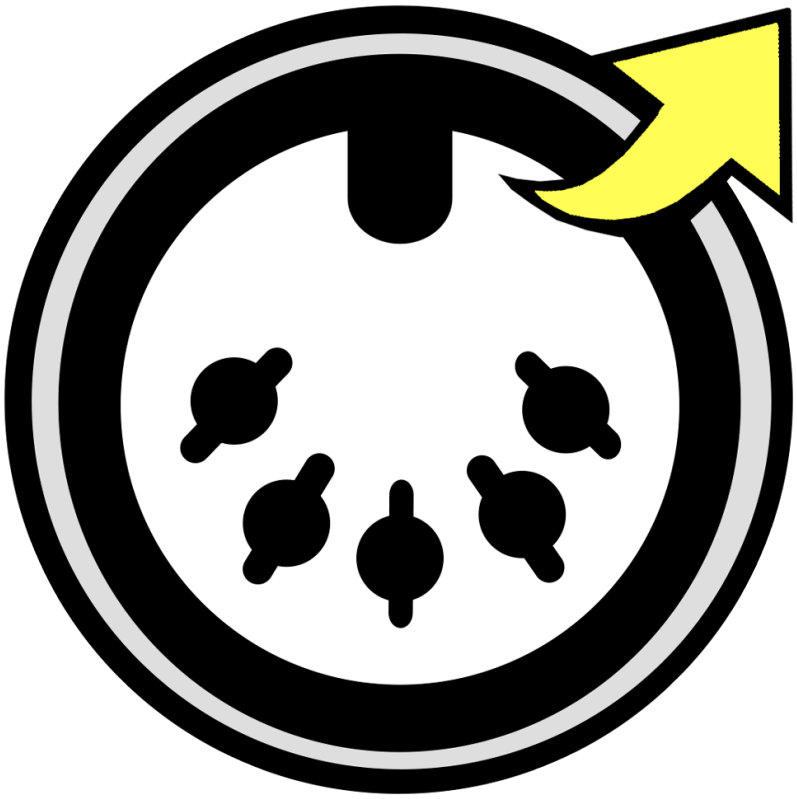
MIDI Router



Hπ INSTRUMENTS

Aaron Andrew Hunt

Changes from Previous Documentation

Here are lists of changes for each version of this documentation.

> Please report typos or problems with this text via email to hpiinstruments@zentral.zone

Current Version, v2 — 15. February 2021

- **Chapter 4:** added and / or reworked sections concerning ***Creating New System Exclusive Events***, ***Testing System Exclusive Events*** and ***Live MIDI Testing***.

Previous Versions

v1 — 22. January 2021

- **Initial Release**

Introduction

MIDI Router MIDI Router for macOS is a utility for routing MIDI communications and monitoring MIDI traffic on your Mac. MIDI data can be transformed between inputs and outputs for practical tasks like reassigning devices to specific MIDI channels, altering note velocities, filtering messages, etc. You can also write scripts which not only alter data but also create new MIDI events, to do anything from converting CC to sysex, to building your own interactive sequencers, auto-accompaniment engines, and even games. A monitor window lets you check up on the data being sent between inputs and outputs, (very useful in troubleshooting situations) as well as custom messages from your scripts. An optional menu item is added to the macOS system menubar, so you can leave the app running in the background and check in easily no matter what else you might be doing.

Bug Reporting & Feedback

Please report any problems you may experience with MIDI Router directly by using the menu item ***Report a Bug***. Before doing so, please also check the MIDI Router reports webpage, which lists all known issues and feature requests. Feedback which is not about bugs may be sent by email directly or using the menu item *Send an Email*.

> Please report bugs as described, and support will proceed via email to resolve the issue.

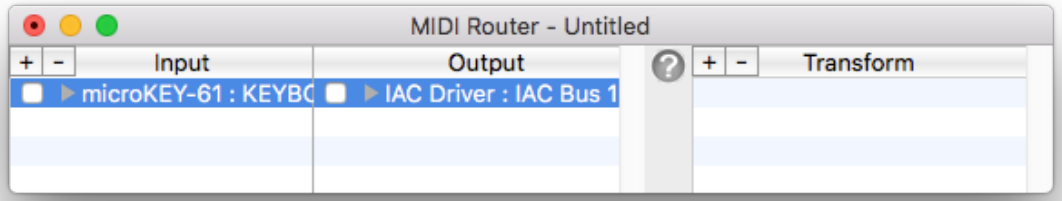
Feature Requests

If the software does not do something you would like it to do, and you are willing to pay for the feature you want, use the menu item ***Request a Feature*** to describe the feature and make an initial offer to pay for it. A professional wage for programming is not expected. 20 € is an acceptable starting point (adding any feature requires several hours of work). If your idea makes sense and your offer is reasonable, then a payment schedule is agreed upon and a testing stage begins. Once testing is done and the feature is verified as working, a new version of the software is released including the new feature(s).

> Not all features are possible or will be considered relevant for the majority of users. A minimum offer of 20 € is standard for all Feature Requests.

1. MIDI Routing

The most basic function of ***MIDI Router*** is to connect MIDI devices, both real (existing as hardware) and virtual (existing only in software), so that messages sent from any input source will arrive at any desired output destination.



Input Sources & Output Destinations

A MIDI Source is also called an Input, referring to something which generates MIDI messages. A MIDI Destination is also called an Output, referring to something which receives MIDI messages. Routing MIDI messages simply consists of connecting an Input Source with an Output Destination, to send data from point A to point B.

MIDI Ports

A MIDI Port can be either an input or an output. Each port supports 16 MIDI channels.

Virtual MIDI Ports

MacOS allows applications to create their own Virtual MIDI Ports which behave the same as a hardware MIDI port.

IAC Bus

MacOS allows you to create your own Virtual MIDI Ports using the utility ***Audio MIDI Setup***. Your Mac should already have at least one IAC Bus created, but you can create more than one. To do so, follow these steps:

1. Open ***Audio MIDI Setup*** from the Utilities folder.
2. Select the menu item ***Window*** ▸ ***MIDI Studio***.
3. Double-click the ***IAC Driver*** icon.
4. Check "***Device is online***".

You may also rename the virtual ports you create using the IAC driver.

Adding a MIDI Route

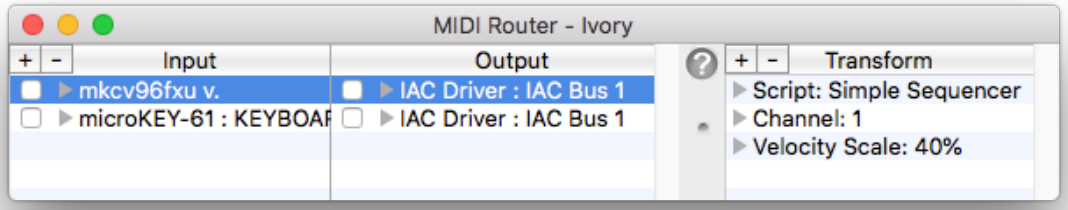
Click the “plus” (+) button above the left list to add a new MIDI Route. Then click the grey disclosure triangle next to “Select Source” to select an Input Source from the popup menu. Click the triangle next to “Select Destination” to select an Output Destination from the popup menu. If the menus are not populated, there are no ports available (see above to create your own virtual ports).

Deleting a MIDI Route

To remove an exiting MIDI Route, simply select it in the list and either press the delete key on the computer keyboard, or click the “minus” (-) button above the MIDI Routes list.

2. Transformations

You may want to transform MIDI data sent from a given Source so that it arrives in some desired form (or is prohibited from arriving) at a Destination. Basic transformation are listed below, and many more complex options are available using a Script (**Chapter 4**).



Filters

The Filter option lets you prohibit certain messages from arriving at the connected Destination. Filtering is available for the following messages: Continuous Controllers (per controller number), *Patch Changes (all)*, *Pitch Bend (all)*, *System Common* messages (all, including *System Exclusive*), and *System Realtime* messages (all).

Channels

Source events sent from any channel can be forced to output on a desired channel. This is especially useful for MIDI controllers which do not allow their sending or receiving channels to be changed in hardware.

Notes & Velocities

Notes can be transposed by any value up to five octaves up (select a positive value) or down (select a negative value). Note ***velocities*** can be transformed in the following ways.

Transformation	Resulting Velocity at the Destination
Fixed	***Constant Value***
Add	***Higher Values***
Subtract	***Lower Values***
Scale	***Higher (> 100%) or Lower (< 100%) Values***

Continuous Controllers

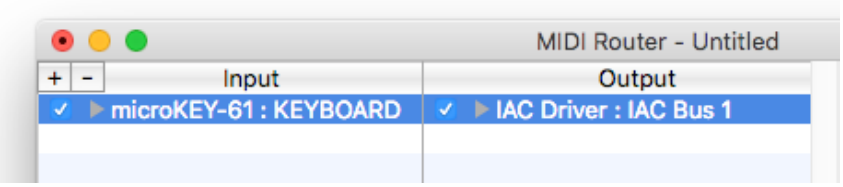
Controller data values may be either filtered completely or transformed in the ways outlined above for Note Velocities.

Scripts

The above options should be adequate for most use cases, but if your needs are more involved or you want to be more creative, you can write your own transformations, including creation of new MIDI events, using Scripts (**Chapter 4**).

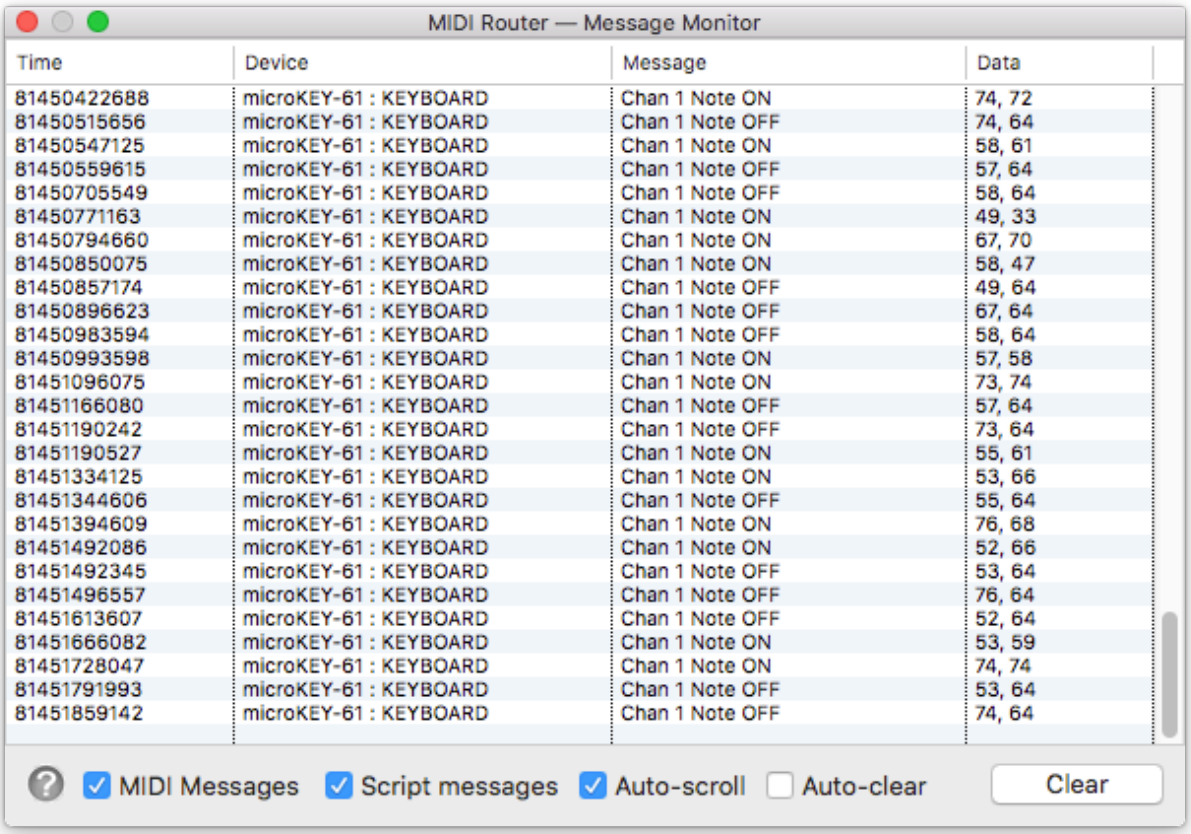
3. Message Monitor

Check a checkbox next to a Source or Destination in your MIDI Routes list to see the MIDI data being passed or transformed between the selected Sources and Destinations.



Appearance

All simple MIDI messages will appear in the list as a status byte followed by two data bytes. In the case of messages which have only two bytes, a third byte may be shown if added in software. Sysex messages are shown as a block of hex data.



Clear, Delete, & Copy

In addition to clearing the list, you may select messages in the list and type delete on the keyboard to remove them. You may also copy messages from the list using the menu item ***Edit*** ▸ ***Copy*** or the keyboard shortcut ***Command - C***, to paste into a text editor, which can be useful for troubleshooting.

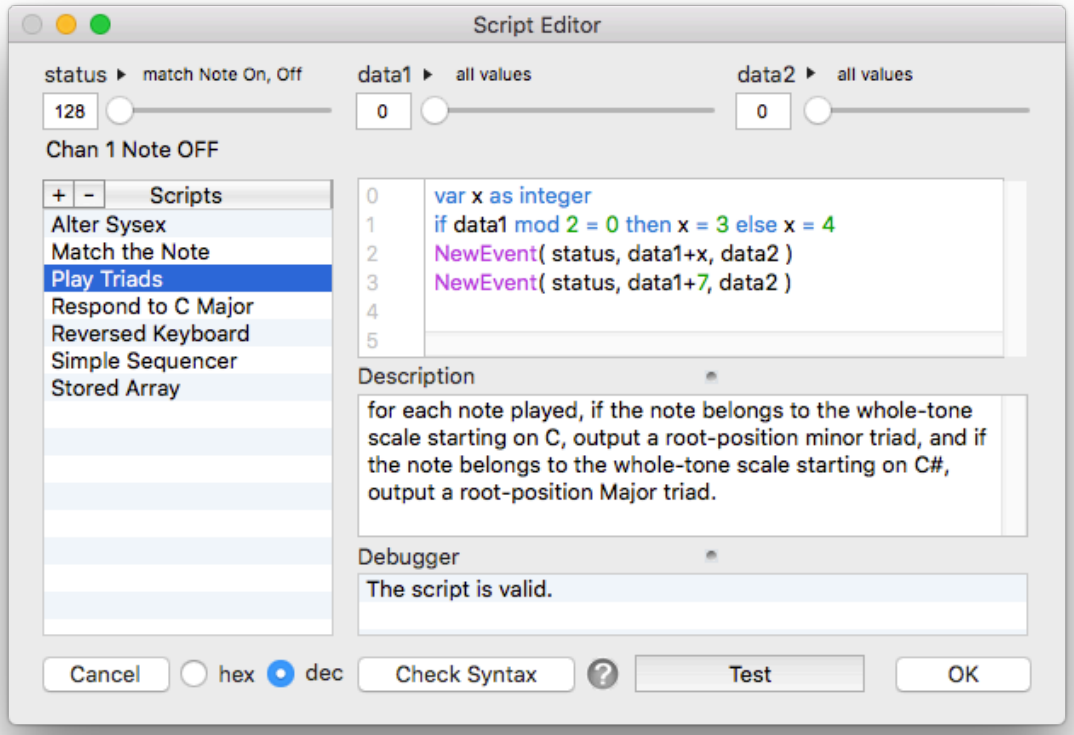
Speed of MIDI Traffic

The monitor window is intended for checking connections, getting feedback from scripts, and general MIDI troubleshooting. The message list is populated using a timer in order not to cripple the performance of MIDI Router while the monitor window is open, but opening the message list may still slow down MIDI traffic, so it should not be left open during normal use.

> Use the Message Monitor for verification and troubleshooting. It is not recommended to leave this window open during normal use, as it may slow down MIDI traffic.

4. Scripts

MIDI Router lets you manipulate incoming MIDI data at the byte level (including **System Exclusive** events) and produce new MIDI messages using a powerful Script Editor. Familiarity with programming concepts and the MIDI 1.0 specification are helpful.



Scripting Language

The core of the scripting language is called ***XojoScript***, belonging to the programming tool used to create this software, a development environment called **Xojo**. At the time of writing this documentation, a users guide of the scripting language is available online at https://docs.xojo.com/UserGuide:XojoScript_Language A language reference menu is also available within the Script Window by right-clicking in the code area. Several example scripts are also included, with commented code to get you started.

Opening the Script Editor

The script window can be opened from the System menu icon under **Script Editor**, from the application menu under **Window ▸ Script Editor**, or from the Transformations list of a project by clicking on an item ▸ and choosing **Scripts ▸ Open Script Editor**.

Interface Basics

The script window includes a list at left for managing scripts, controls at the top for selecting input conditions which will trigger your script, a field at the upper right for typing in your code, and a smaller field below that for typing in an optional description. A list for **Debugger** output provides information on scripting errors when you click **Check Syntax**, and displays both pseudo- input and output data when you click **Test**. Clicking the **Cancel** button will close the window and discard all the changes you have made. Clicking **OK** will save all changes. Byte values can be displayed either in hexadecimal (**hex**) or in decimal (**dec**), as needed.

About Bytes and Channels

MIDI byte values range from 0-127 for data bytes, and 128-255 for status bytes. Although it is common to refer to MIDI channels as 1 through 16, within your script any methods using the keyword Channel are zero-based, so the first channel is always channel 0, not channel 1.

Managing Scripts

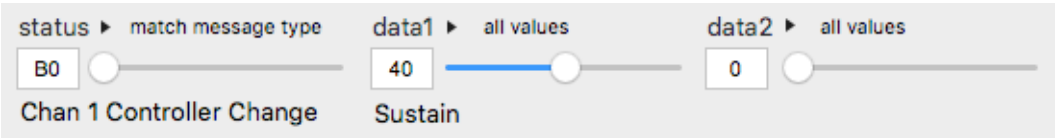
The list at left titled **Scripts** lets you add, remove, rename, and duplicate your scripts. Click the plus button (+) to add a new script. Click minus (-) or type Delete to remove a script. ***Double-Click*** a script to rename it. To duplicate a script, ***Right-Click*** or ***Control-Click*** a script and select Duplicate from the popup menu. Script names must be unique. Scripts are stored as text files and are stored internally in the following location:

/Users/UserName/Library/Application Support/MIDIRouter/XML/

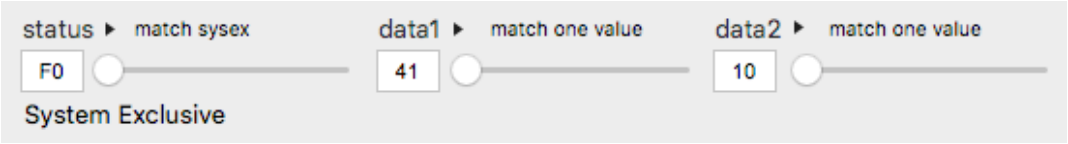
If you purchased MIDI Router from the Mac App Store, the above location exists inside your /Users/Username/Containers directory.

Selecting Input Conditions

The controls and menus at the top of the script window correspond to three bytes of an input Event: status, data1, and data2.

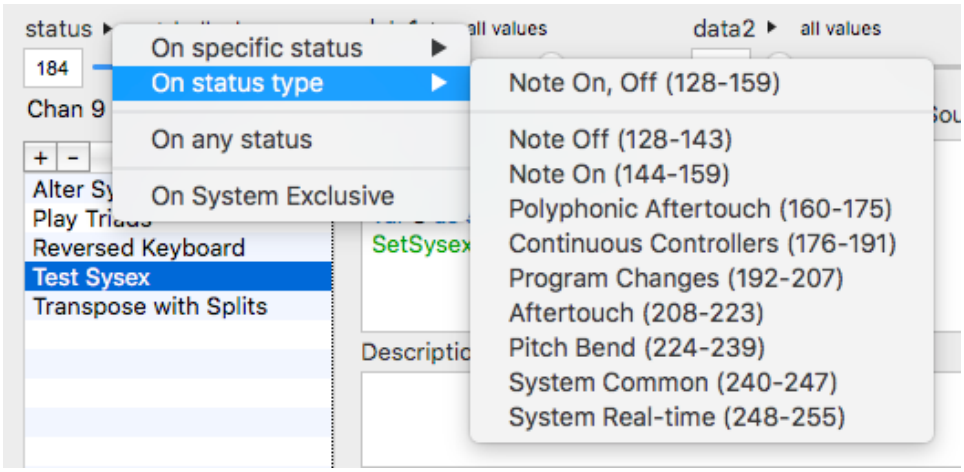


Clicking each small triangle ▸ will open a popup menu showing various value-matching options. A description of the currently selected status value is shown below the status slider. When the status is set to a **Controller Change** byte, a controller description of the data1 value also appears. When the status is set to **match sysex,** then the bytes data1 and data2 correspond to the second and third bytes of an incoming sysex message. For example, the following settings match the first three bytes in hex of a sysex sent by a Roland keyboard.



Status-byte Input Options

When a new script is added to the list, its status condition defaults to respond to all incoming Note messages. Click the triangle ▶ to assign a different status condition.



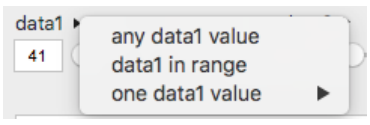
Note that is possible to match the **System Exclusive** status byte in several different ways:

1. **On specific status ▸ System Exclusive**
2. **On status type ▸ System Common**
3. **On any status**
4. **On System Exclusive**

Above, options **1** and **4** are the same, meaning your script will execute *only* when a sysex message is received. The dedicated menu item at **4** is simply a shortcut. Options 2 and 3 mean your script will respond to either the *group* of **System Common** messages which includes **System Exclusive**, or to *all* messages, which also includes **System Exclusive**.

Data-byte Input Options

It is possible to match each data byte as: ***any value***, ***in range***, and ***one value***.



If you set the status condition to match a **Controller Change** byte, the menu under ***one value*** will populate with a list of controller names. Selecting the ***in range*** option begins a two-step process to select the range:

1. move the slider to a lowest value, then click [**select lowest value**]
2. move the slider to a highest value, then click [**select highest value**]

After you have selected lowest and highest values, moving the slider sends only values within that range. If you select the **one value** option, the slider will be locked to a single value.

Altering or Filtering the Input Event

Your script is executed once per input Event, based on the status and data byte conditions you choose. An input Event is either a simple 3-byte MIDI message or a complete block of sysex (System Exclusive) bytes. You can alter a 3-byte input Event within your script by assigning new values to status, data1, or data2, and those new values will be passed to the Destination unless you suppress the input Event using FilterInput. The content of an incoming sysex block can also be altered using SetSysexByte(index, byte) and SetSysexBytes(index, bytes()), or filtered using FilterInput. See further details under ***System Exclusive Events*** below.

Storing Input Events

Input events can be stored from one iteration of the script to the next using StoreEvent, useful for matching input patterns consisting of multiple events. The events are stored in a simple array. To get the size of the array, call StoredEventSize. Look up the properties of stored events using StoredEventTime(index), StoredEventStatus(index), StoredEventData1(index), and StoredEventData2(index). A convenience method StoredEventChannel(index) is also available. The array can be emptied at any time using InitStoredEvents.

Creating New Output Events

Any number of new output Events can be produced using NewEvent(statusByte, dataByte1, dataByte2), and new sysex blocks can be sent using NewSysex(bytes()). Note that the data type for the array sent to NewSysex must be bytes() and not integers(), or the debugger will throw an error. The new events are sent immediately to output when the script executes.

Getting Event Times

At any time you can call the function System.microseconds to get a Double value for the current time. The **Simple Sequencer** example script shows how this can be used to calculate the on and off times of incoming note messages.

Creating Time-Delayed Output Events

New Events can also be sent with a time delay in milliseconds, using NewDelayedEvent(msDelay, statusByte, dataByte1, dataByte2). A convenience method is included for handling both the On and Off event of new delayed notes using NewDelayedNote(msDelay, msDuration, channel, note, velocity). Passing zero for the msDelay parameter will cause the event to be sent out immediately. In the case of NewDelayedNote the note will turn itself off after the specified msDuration. Note that these times are in milliseconds, so when using System.microseconds you will need to convert values accordingly.

System Exclusive Events

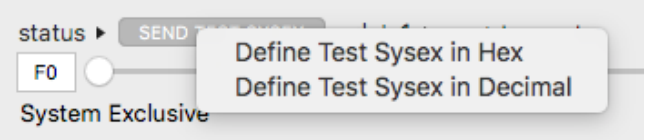
A sysex event consists of a block of bytes of any size beginning with **F0** (240) and ending with **F7** (247), with only data bytes having values **00**-**7F** (0-127) in between. To get the size of the block, use GetSysexSize. The number returned is the total number of bytes in the block, including the start and end bytes. To look up byte values anywhere in the block, use GetSysexByte(index) or GetSysexBytes(index, size). The first byte is numbered zero (0). You can alter byte values using SetSysexByte(index, byte) or SetSysexBytes(index, bytes()). Unless you call FilterInput, the block will be sent to output.

Creating New System Exclusive Events

There are two ways to create a new sysex block. You can either create an array of bytes using data type Byte or UInt8 and then call NewSysex(bytes()), or you can pass a string of bytes as a string using NewSysex(“F0 F7”) where your data bytes are listed between F0 and F7. Examples of both methods are given in the Script window. New blocks will be sent immediately when the script executes. Keep in mind that no error checking is done on sysex blocks you create; you are solely responsible for their structure. Likewise, if you alter the structure of an incoming sysex message, you need to make sure its byte structure remains valid.

Testing System Exclusive Events

If your script is listening for sysex events, you can define a test sysex message to send to your script. First click **Test** at the bottom of the window, then Right-Click the SEND SYSEX button which will appear at the top left of the window, and select an option from the popup menu.



A window will appear in which you can type in a list of bytes. Use the ***asterisk*** character (*****) to designate any data bytes which should be randomly generated. In this case your sysex message ***is*** checked for validity, so that the start and end bytes are sure to be correct and all bytes in between are values between 0 and 127. The list of bytes you enter will then be sent to your script when you click SEND SYSEX during testing.

Storing & Recalling Values

Local variables you create in your script are only in scope during a single execution of your script, but you can also store values from one iteration of your script to the next using special functions for that purpose. Since MIDI is based on bytes, usually you will be working with bytes, but in modern programming, integers are the default data type, so the default type for storing and recalling is an integer. Integers handle many more values than bytes, including negative values, which are often used as flags when programming, so in most cases they are more useful than working only in bytes. The keyword Store(“Name”, myInteger) stores an integer value, where “Name” is any arbitrary string you want to assign, and myInteger has been created as an integer using the var keyword. Other data types work the same way, with methods named for each datatype. To recall a stored property, use the keyword Recall(“Name”) along with the data type if it is not an integer.

```
Store( “Name”, myInteger )
StoreDouble( “Name”, myDouble )
StoreBoolean( “Name”, myBoolean )
StoreString( “Name”, myString )

var myInteger as Integer = Recall( “Name” )
var myDouble as Double = RecallDouble( “Name” )
var myBoolean as Boolean = RecallBoolean( “Name” )
var myString as String = RecallString( “Name” )
```

The example scripts follow a stored value naming convention, where the “Name” property is the same as the locally named variable (or the same but with spaces added). This is just a logical convenience; the local name and the stored name do not in fact have to match.

Storing & Recalling Arrays

To work with data lists from one script iteration to the next, the script language includes similar store/recall methods as those explained above for arrays. Unlike single values, arrays are internally stored as real one-dimensional arrays of the data type specified, so unique methods are required both to store and to recall the arrays. The store and recall methods are as follows.

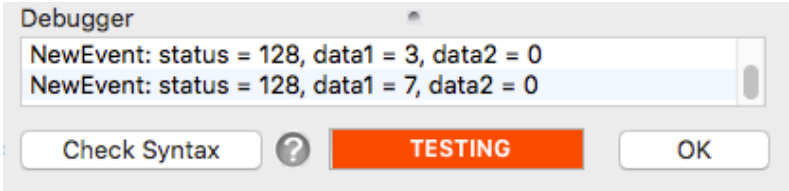
```
StoreArray( “ArrayName”, myIntegerArray() )
```

```
StoreDoubleArray( “ArrayName”, myDoubleArray() )
StoreBooleanArray( “ArrayName”, myBooleanArray() )
StoreStringArray( “ArrayName”, myStringArray() )

var myIntegerArray() as Integer = RecallArray( “ArrayName” )
var myDoubleArray() as Double = RecallDoubleArray( “ArrayName” )
var myBooleanArray() as Boolean = RecallBooleanArray( “ArrayName” )
var myStringArray() as String = RecallStringArray( “ArrayName” )
```

Checking Syntax & Testing Scripts

Click **Check Syntax** to compile your script and output any error messages to the Debugger. Problems in your code will be marked with line and character numbers, with red dots placed at the left of your code by the line number, showing you exactly where errors exist in your code. When you click the **Test** button, it will read **TESTING**.



While testing, you can move sliders or click the status button to send pseudo-messages to your script input to verify that your script is working as intended. Note that actual MIDI data is *not* involved when testing. To end testing, click the **TESTING** button again.

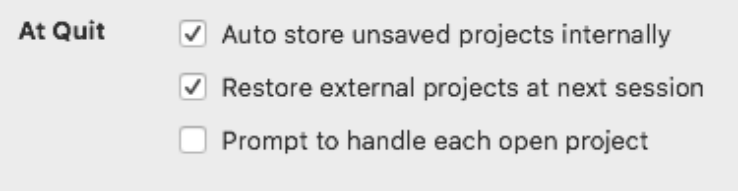
Some errors will not appear in the debugger and will only appear at runtime when the code is actually executing. Beware that in some cases code that you have written containing a bug such as an infinite loop may cause MIDI Router to crash. Your script is however never fatal for the application itself, only perhaps for a given session.

Live MIDI Testing

To test your script in real time with actual MIDI input and output, assign the script to a MIDI route in a project window and then edit the script. Changes you make to your script code are updated in real time as you type, so if you are working on a live MIDI route, you can test with real live MIDI input as you go. On the other hand, because live input can sometimes produce unexpected results when your code is incomplete (possibly resulting in a crash), you will want to make sure MIDI input is not being sent to the script while you are editing it.

5. Files

In the ***Preferences Window*** you will find a list of options for the behaviour of MIDI Router when you quit the application.



Auto store unsaved projects internally

Check this option if you would like open projects that have not been saved to be stored automatically when you quit MIDI Router. You can then work in MIDI Router without having to manage external project files at all, which may be more convenient in some cases. The project files will instead be stored in the following location:

/Users/UserName/Library/Application Support/MIDIRouter/Sessions/

If you purchased MIDI Router from the Mac App Store, the above location exists inside your /Users/Username/Containers directory. Note that project files which have been opened as external files will *not* be stored internally, but rather will be stored at the location of the external file.

Restore external projects at next session

Check this option if you would like all open projects to be reloaded automatically the next time you open MIDI Router. MIDI Router will then store the paths to open projects in a file called ***projectpaths.xml*** in the directory listed above. When using this option, be aware that if you move your project files around after quitting MIDI Router, the paths will no longer match and the moved files will not be opened when you reboot MIDI Router.

Prompt to handle each open project

Check this option if you want to use the above automatic project saving and loading options, but also want to have control over exactly what will happen to each project when you quit MIDI Router. A dialog will then appear for each window asking you what you want to do with that project. This is the default behaviour.

Discarding a Project

When using the automatic options described above, MIDI Router gives you options to save projects internally or externally. If you want to discard the file, select ***Options***, and select ***Discard***, or choose ***Save Externally***, cancel that operation, and then close the window.

Credits

All versions of MIDI Router are designed and programmed by Aaron Andrew Hunt, using Xojo and MBS Plugins on a Mac.

This documentation is written by Aaron Andrew Hunt.

Thank you for supporting H-Pi Instruments and MIDI Router.