

Actions

This menu item has the following choices:

Stop or Start

Clear

Clear Status

Panic

Send

Run REXX

Stop WScript

Keyboard

Play MIDI



Stop/Start

This menu or toolbar command toggles starting or stopping sending MIDI data to the monitor window.



Clear

This menu or toolbar command clears the current display window.



Panic Stop

This menu or toolbar command turns off all notes and controllers on all channels.

Clear Status

Clears the Status Window.

Send

This is a quick way to send one of the three standard reset Sysex strings. You can also select a Sysex file to send.

Stop WScript

This menu command provides a way to signal a script or other COM object using the COM Scripting interface that the user wants to terminate.



Play MIDI

This menu command starts the external MIDI Player you have selected under Options | Select MIDI Player. By default a small MIDI Player that uses the services provided by Windows (MCI) called MidiBar is selected. MidiBar is provided free of charge with the MIDI-OX installation. You can choose to run any executable file, however.



Keyboard

To enable the Keyboard select the Keyboard item under the Actions Menu or select the Keyboard icon on the toolbar. The toolbar icon will "flash" while the keyboard is enabled.

You can use other windows while the keyboard is "on" but remember that most keyboard keys are being trapped. You will have to use the mouse for some things that you could normally do with the keyboard.

[Playing Notes](#)

[Changing Channels](#)

[Playing Chords](#)

[Changing Patch Number](#)

[Mode 1](#)

[Pitch Bend](#)

[Mode 2](#)

[Modulation](#)

[Mode 2 Key](#)

[Sustain](#)

[Changing Octaves](#)

[Panic Button](#)

[Changing Velocity](#)

[Keyboard Mapping](#)

[Arpeggiator](#)

Playing Notes

The keyboard keys are mapped to mimic a piano keyboard. The row beginning with "Z" is the bottom octave starting with a note of C and the row starting with "Q" of QWERTY fame is one octave higher. See [Changing Octaves](#) for more details on selecting the exact octave you want. The "black" keys are on the row above these two rows, so some of the keys in those rows will be silent as they don't map to a black key. Multiple keys can be played at once to create chords. Note that there will be a limit to the number, and sometimes combination, of notes that most computer keyboards can play at once.

Playing Chords

There are two Modes for playing chords:

Mode 1 is turned on by selecting **Caps Lock**.

Mode 2 is turned on by selecting **Scroll Lock**.

See descriptions for [Chord Mode 1](#) and [Chord Mode 2](#).

Chord Mode 1

Selecting Caps Lock selects Chord Mode 1. All keys will play a chord rather than a single note. The type of chord can be selected from the Extended Keypad (make sure Num Lock is on). All chords designated with the word Toggle will be active once the key is pressed until the key is pressed again, or the Clear Toggle key is pressed. This means that you can play combinations of toggles.

Numpad 0	Major
Numpad 1	Inverted Toggle - base note up an octave
Numpad 2	Second Inversion Toggle - fifth down an octave
Numpad 3	Minor
Numpad 4	Minor Sixth
Numpad 5	Minor Seventh
Numpad 6	Sixth
Numpad 7	Seventh
Numpad 8	Octave Toggle - base note repeated an octave higher
Numpad 9	Ninth
Numpad /	Diminished Toggle
Numpad *	Augmented Toggle
Numpad .	Clears All Toggles

Chord Mode 2

Selecting Scroll Lock selects Chord Mode 2. The keyboard plays single notes as it normally does while the NumPad keys 1 through 8 play one of 7 diatonic chords based on the selected key. (See [Change Key](#).) The seven chords are:

I	maj7
II	min7
III	min7
IV	maj7
V	7
VI	min7
VII	min7b5
VIII	maj7 (octave)

There are 3 different chord inversion "flavors":

- 1 Inverts all the chords to minimize voice leading.
- 2 Is a straight non-inverted block chording.
- 3 Uses a 'drop 2' inversion: the 2nd highest voice is dropped an octave.
This gives some spread to the chords.

Pressing the Numpad Multiplication key (*) iterates through the inversion types. In addition the Numpad Divide key (/) will toggle the octave range up and down one octave for the chords.

Change Key (Mode 2)

To change the Key that chords play in while in Mode 2, hold down the Control key and select one of the Function keys F1 through F12. F1 is the key of C and is the default. The other keys move up in half steps in the same order as a piano keyboard.

Changing Octaves

The F2 through F9 keys select the range of octaves from lowest to highest. The default setting is F6. At this setting the "Q" key is set to MIDI Note 60, commonly considered Middle C.

Changing Velocity

The Page Up/Page Down keys change the velocity of the notes. The default setting is 100 and each key press adds or subtracts 4.

Changing Channels

The Minus (-) and Plus (+) keys on the extended Keypad change the channel the keyboard plays on. The default is Channel 1.

Changing Patch Number

Hold down the Tab key and use the NumPad keys to enter the desired patch number -- based from 0 to 127 (be sure to turn on Num-Lock). When the Tab key is released, the patch change is sent. If you release the Tab key without entering any numbers, the default patch change is sent, patch number 0. To clear an entry and start over, with the Tab key still held down, push the Decimal (NumPad Delete) key and then enter the correct numbers. If you pushed the Tab key by mistake or for some other reason changed your mind about sending a patch change, press the Decimal key and release the Tab key. No change will be sent.

Pitch Bend

The Insert and Delete keys send Pitch Bend messages with MSB increments of 8. Because this is changing the MSB of the pitch bend message, the actual number of pitch bend steps changed by each key press is 1024. There are 8192 increments up or down (as defined by the MIDI spec), so 8 presses on the up arrow bring you up to the maximum, for instance, or 8 presses down to the minimum. Either key can be held down to send repeating messages at the rate set by your system Repeating Key setting. To instantly center Pitch Bend, Press the END button.

Modulation

The Left and Right arrow keys send Modulation Controller messages (Controller 1) in increments of 16. Holding down either key sends repeating messages.

Sustain

Holding the SHIFT key down turns on sustain by issuing a MIDI Pedal controller event (Controller 64) with a value of "on" (127). Releasing the SHIFT key sends the same controller with a value of 0, turning sustain off.

Panic Button

Pressing the HOME key causes a Reset All Controllers Message, All Notes Off Message, and turns off sustain. This only affects the current channel, not all channels.

Keyboard Mapping

Note that one of the mapping options is to enable mapping the computer keyboard. You could map the Pitch Bend or Modulation behavior to something else, like Pan, for example. You could also map a particular note (or range of notes) to perform some other command, like a patch change, or to turn on Reverb, etc. See [Data Mapping](#) for more information.

Arpeggiator

To play arpeggios in either chord mode, press the "A" key. When you release it you will be in Arpeggio mode. To adjust the delay time between notes, hold down the "A" and the CTRL key while selecting the time on the Extended keypad. When you release the "A" key (while the CTRL key is still down) the new delay time will take effect. The delay time is in milliseconds.

The table shows how long a beat lasts in milliseconds for various Beat per Minute settings, and also shows calculations for note lengths at 2, 3, and 4 notes per beat. The formula used to calculate the beat length is: $\text{BeatLength} = 60 / \text{BPM} \times 1000$.

BPM	Beat Length	2 Notes/Beat	3 Notes/beat	4 Notes/Beat
80	750	375	250	188
100	600	300	200	150
120	500	250	167	125
140	429	215	143	108
160	375	188	125	94
180	333	167	111	84
200	300	150	100	75

Note This timing is not controlled by a low level timer and thus is not rock-solid. The fewer other things the system is doing, the better the timing will be.



Control Panel

This window allows you to send various Controller and other messages. It allows you to send MIDI events to both the display, and MIDI output. By using it you can send Patch changes, volume, pan etc. in real time. It is a non-modal dialog box, which means you can keep it open all the time the program is running.

Bank / Patch Panel

The **Bank / Patch Panel** allows the sending of bank and program changes to synthesizer devices.

Bank Section

The Bank section consists of the controls in the upper left of the dialog. There are two check-boxes, two edit controls and a **send** button. There are several different methods that synthesizers use to respond to Bank change directives. You will need to determine which is supported by your gear. MIDI-OX supports three methods (Normal, MSB only and LSB only).

The **Normal** method is to send a Controller 0 message, containing the *least significant byte* (LSB) of the bank change number; followed by a Controller 32 message, containing the *most significant byte* (MSB); followed by a Patch change message. This method is selected by checking both **MSB** and **LSB** checkboxes. The bank change will be sent on the current channel selected at the bottom of the dialog (or all channels if that option is set).

The **MSB only** method sends only the Controller 32 message followed by a Patch change message. The MSB method is selected by checking only the MSB checkbox.

The **LSB only** method sends only the Controller 0 message followed by a Patch change message. The LSB method is selected by checking only the LSB checkbox.

The spin-button edit controls obey the current **auto-send** setting: if it is checked, incrementing or decrementing a value causes the bank change to be sent. The Bank section has its own **Send** button to avoid confusion and ambiguity with the controller list send button. Pressing the Bank send button will send the currently numbered bank followed by the current patch number for the channel. If the **Send All Channels** button is checked, the bank/patch change will be sent on all 16 MIDI channels.

Patch Section

The button interface is reminiscent of Roland synthesizer gear. There are two sets of radio buttons: the top button rows (numbered 1-16) select the instrument **section**; the lower button row (numbered 1-8) selects individual instruments within a **section**. In addition, a true patch number (1 based) is displayed in the upper right corner. Using all the buttons there are 128 possible instrument selections from 1-1 through 16-8. A program change is sent as soon as one button from each set (section, patch) is enabled.

The buttons will update to reflect an externally generated program change for the current channel. Patch settings are remembered for each channel.

Controller Selection

You select the controller to send from the drop down combo-box, the channel from the edit control. If the **Auto Send** Value box is checked, the events are sent as soon as the value changes. If the **Send All Channels** button is checked, a value change will be sent to all 16 MIDI channels. You can use the up and down arrow keys (as well as page-up and page-down) to change the Spin controls (in addition to clicking on the up or down arrowheads).



MIDI Devices

The MIDI Devices dialog allows you to attach your MIDI Ports and configure their routing. Once you are satisfied with a particular configuration, press **OK** to attach the ports and close the dialog. Pressing **Cancel** leaves things the way they were before you entered the dialog.

Concepts

Input Ports supply MIDI data to MIDI-OX. The originating device is usually a hardware port connected to a MIDI Keyboard, Guitar, or Wind controller; but it can also be a *virtual* MIDI device such as MIDI Yoke, that supplies the MIDI output from another application.

Output Ports pass MIDI data from MIDI-OX to Internal or External synthesizers. Again, a *virtual* output port (MIDI Yoke) can pass data to another application.

MIDI-OX allows multiple input and output ports to be opened at a time. You don't always want all of the MIDI data from all of the inputs passed to all of the outputs, so we have added a *routing* mechanism that allows you to specify which output ports receive various types of data.

Routing

Each time you click on an input port in the upper left listbox (MIDI Inputs), it selects a device for MIDI-OX to open and listen to. A *Port Map Object* is also added to the lower right Tree View (a little green MIDI port icon followed by the Port Name) that can be used when configuring the upper right (Port Mapping) Tree View.

Each time you click on an output device, it selects an output device and sets up a potential mapping to all selected input devices. A light blue MIDI port icon is added to the Port Mapping view. If you expand this line (click on the '+'), you will see each selected input port (green icon) indented below. Each of the input ports can also be expanded to display which channelized and system data they will pass along to this output port.

To block an entire input port from being routed to this output, select the indented input (highlight it) and then press **Delete**. Alternatively, you can select the object, right-click on it and choose **Delete <item>**. If you change your mind, you can drag the input port from the **Port Map Objects** view and drop it on top of the output port name.

To choose what channelized data is supplied to an output port, you can expand the input port and delete individual channels. It is important to note that you are only specifying the channel routing for *this* output port. All channels of data from the input port will still be supplied to any other open output port.

You can also specify what type of system (non-channelized) data should be passed from an input to this output. To prevent the passing of System exclusive data, expand the input port, and then expand the **System** line and delete the **System Exclusive** line. You can always add it back by dragging the **System Port Map Object** from the lower right view, and dropping it on the Input port in the **Port Mapping View**.

In addition to normal input ports, there is a virtual **MIDI-OX Event** object (dark green icon). This special port mapping represents events that originate in MIDI-OX. For example, to block System Exclusive data generated from the SysEx View from being sent to a particular output port, expand the output port, expand the MIDI-OX event object, expand the System object and delete the System Exclusive line.

Presets

Once you have setup and configured a routing, it can be saved as a preset. To do this, type a name into the **Presets** combo box (in the lower left), and then press the **Disk** icon. Presets can be restored by choosing the name from the Presets drop down list. You can delete the displayed preset by pressing the 'X' icon.



Selected Devices

This command causes the names of the current devices to be printed out in the monitor window.

File Menu

The File Menu in **MIDI-OX** addresses a few file oriented actions.

Close

Log

Load Profile

Save Profile

Exit

Close

Closes the currently active window. If the active window is the **Output Monitor**, it is merely minimized because it always remains open.



Log...

Choosing **Log...** brings up the Logging dialog. Here you can enable and disable logging, and choose what type of session log to create. You can also view an existing log file.

Log Enabled

If you check this box and press **OK**, a session log is started. When logging is on, the Log menu option will have a check mark besides it. To stop logging the session, uncheck this option and press **OK**.

Format

Normal Format. The log mirrors the monitor display.

'MIDI to Text' Format. When this option is selected a special output format is generated. The resulting log can be input to the **T2MF** program to generate a Standard MIDI File consisting of the logged performance. See [Logging](#) for more information. This option also chooses and disables the **Overwrite Log** radio button and disables the **Append to Log** option because appending to an existing log file would make the output useless as a **Text to MIDI** format.

Raw MIDI Message Format. The MIDI data is dumped to the log. Each full message is separated by a space.

Display Dialog Warnings

Checking this option causes a warning message to be displayed when a log file is open and the **View Log...** button is pressed. If you continue the Log is closed, and you have to exit and restart the dialog to restart logging. This is necessary because files can not be viewed when they are opened for logging.

Append to Log

This is the first of a pair of radio buttons. Choosing this option causes the output to be added on to the end of an existing log file.

Overwrite Log

This is the second radio button. Choosing this option causes the logging session to overwrite an existing log with the same name. This option becomes the only choice when **Use 'MIDI To Text' Format** is selected, because the **Text to MIDI** converter expects exactly one entire MIDI file specification to be supplied.

View Log...

The **View Log...** button launches the text document. If a viewer is with the document associated (normally **Notepad** or **Wordpad** in Windows), the Log file is opened. Be sure to use the **.TXT** suffix when naming a Log to successfully use this feature.

[...]

The **[...]** button means **browse**: it allows navigation of the hard drive to choose an appropriate location for the log file.

Exit

Choosing Exit, closes and ends the MIDI-OX program session.

Profiles

Profiles allow you to set up MIDI-OX the way you want and save all the settings to be loaded again at a later time. MIDI-OX already will re-load the last settings, but profiles allow you to store several setups and quickly change between them.

Initialization Profiles can be specified on a command line, or added to the command that is launched via an icon. The general form for a command line profile is:

Midiox.exe D:\yourpath\profile.ini

If you use just the filename (example: **profile.ini**), MIDI-OX looks in the *current directory* for the file. If it finds the file it will load the profile and place the profile filename on the application caption bar. If the profile name is *not* on the caption bar, MIDI-OX was not able to load it. Because MIDI-OX allows only a single pathname, you don't have to quote long pathnames containing spaces, but it should handle it correctly if you do. If you're making a Windows shortcut you need to quote the long pathname to the executable, or use the 8.3 form. Examples:

"C:\Program Files\MIDIOX\Midiox.exe" D:\path\profile.ini

C:\PROGRA~1\MIDIOX\midiox.exe C:\path name\my.ini

Profiles can also be saved and loaded from within MIDI-OX. Use the **File | Save Profile...** command to create or update a profile. They are ascii text files, that resemble Windows INI files, so they can be viewed and tweaked in a text editor (be careful!). To load a profile while MIDI-OX is running, use the **File | Load Profile...** menu command. For any settings that aren't specified in the profile, MIDI-OX uses the current settings or the default settings.

When each MIDI-OX instance closes, it saves all its current settings to the Windows Registry; even when the program has been launched with a profile. This allows for up to 16 separate default profiles in the registry. When a MIDI-OX instance is run *without* a command line profile, it uses the next numbered default profile in the registry. For instance: when the second instance of MIDI-OX is launched, it obtains its settings from **Profile 2** in the midiox section of the registry. When that instance exits it saves its current settings back to **Profile 2**. For the second and subsequent instances, MIDI-OX places the instance number on the application caption bar.



Filter Window

This window allows you to select which MIDI messages, if any, are filtered from the display. You can also toggle whether or not to strip out the filtered messages from the outgoing data stream.



Help Menu

Standard Help Menu and About box.

Author and Version Information

MIDI OX is Copyright © 1997-2000 by Jamie O'Connell and Jerry Jorgenrud

E-Mail Jamie O'Connell:

jamieOConnell@midiox.com

E-Mail Jerry Jorgenrud:

jorgenrud@attolobal.net

Version Information is available from the **About** command in the Help Menu.

Web Site

The MIDI OX Web Site is located at: <http://www.midiox.com>



Logging

The logging function allows saving of received MIDI information to disk. The standard format for this is identical to the information displayed in the **Output Monitor** view, plus added header and trailer information. There is also a MIDI to text option: this saves output in a format which can be converted into a standard MIDI file; and a Raw MIDI Message option.

MIDI to Text

The **MIDI to text** formatting option saves received MIDI data in a format which is directly compatible with Piet Van Oostrum's **T2MF** program. Check the MIDI-OX web site for program availability.

MIDI-OX assumes and calculates a meter of 120 beats/minute and 120 ticks/beat: you will probably need to edit any resulting MIDI files if you care about the meter. The output specifies a type 0 MIDI file and it starts recording the MIDI data (at tick 0) as soon as the first MIDI message is received at the **Output Monitor**, after enabling logging.



Data Mapping

Invoking this command brings up the Translation Mapping window. The MIDI data mapping function allows the transformation of received MIDI events. It can change any type of MIDI Message into any other type. In addition, channel information and message parameters may also be mapped to other values and ranges, and swapped, extended or cloned. Recently added, is multiple message mapping from and to NRPN numbers and values, as well as support for mapping to SysEx events. The uses for this capability are limited only by your imagination. Some obvious ones include performing keyboard channel splits, mapping controllers from one type (or data range) to another, setting up "hot keys", e.g. mapping high or low notes to patch changes, or even inversion of the entire keyboard (high notes at the left and low notes at the right).

Map Usage

A mapping consists of an input filter specification and an output map. You can have up to 256 mapping items active at once by default (see below how to change this value). As soon as a MIDI message satisfies a mapping item (input filter), the output mapping is performed and the transformation is sent to any open output port and the output monitor window. You can optionally direct that the message be cloned for further mapping, or have it return as soon as a match has been determined.

You can change the order of mappings to achieve priority considerations. Highlight the map entry and use the Up or Down arrow keys to move it in the list. You can also Delete the highlighted entry or Clear the entire map.

Asterisks are used in the channel map to show events that aren't modified:

Asterisks in the input **All** values pass the filter;

Asterisks in the output The input value is transferred to the output unmodified.

In addition to being a shortcut notation, using the asterisks is more efficient. If actual values are specified, the input values are interpolated when calculating the output value. Many times this is useful. For example, note velocities or channel volume messages might be mapped from 0 to 127 on the input to 100 to 127 on the output, to limit their range. The **Value1** and **Value2** parameters are color-coded (blue and red respectively), because they values may be swapped or extended in the map editor dialog.

A check mark is displayed in the **Clone** column where an input message should be duplicated. When a message is *cloned* it continues to be mapped, even after it has already been converted to another message. If the clone reaches the end of the map without further conversion, it is output in it's original form. You can use this feature to pass the original value through the system regardless of whether a copy is mapped.

You can save and load maps. The suggested default extension is OXM, but you can call them whatever you want.

Note In order to use the map you have loaded or defined, the Map Is On checkbox must be checked.

There is an additional checkbox specifying if you want the computer keyboard to be mapped. If this is checked but the Map Is On box is not checked, the mapping will only occur from the computer keyboard. This could be useful for mapping the computer keyboard notes to patch changes, for example (or pan, volume, reverb, whatever) without altering the normal flow of events from input to output.

If they are both checked, then both the input and the computer keyboard will be mapped according to the same map.

For some examples and instructions on creating or editing mappings, see:

[Define Mapping Dialog](#)

[Example 1 \(Keyboard Split\)](#)

[Example 2 \(Reverse Keyboard\)](#)

[Example 3 \(Patch Mapping\)](#)

Changing the Map Items Maximum

Although a large number of mapping items can be assigned in the mapping dialog, the low level memory for this feature is allocated and fixed during program startup. By default, 256 mapping items can be assigned. This maximum can be increased (or reduced) by changing a parameter in **MOXLIB.INI**. This INI file should be located or created in the MIDI-OX application directory. The parameter is located in the **[Options]** section and is named, **MapItems**. If you change the INI file, you must restart the program before the setting takes effect.

Example:

[Options]

MapItems=384 ; Creates room for up to 384 map items

NRPN Mapping

You can map multi-message NRPN number and value to MIDI messages, SysEx or other NRPN numbers. When you choose NRPN as event type, the number and value fields have greater range than normal MIDI values. NRPN numbers and values are represented by 14 bit quantities: 0 - 16383 decimal. This is because 2 MIDI bytes (MSB and LSB) are used to formulate the number. When only MSB is transmitted, the magnitude of the value is the similar (0 - 16256), but the granularity is not as fine. There are several MAP level options available for NRPNs:

[x] Wait for Complete NRPN data entry (Ctrl 38)

When checked, causes MIDI-OX to wait until all 4 messages comprising a NRPN have been received. When unchecked, reacts as soon as Ctrl-6 (Data Entry MSB) has arrived (3 messages total).

[x] Map NRPN data increments (Ctrl 96,97)

When checked, causes MIDI-OX to consider data increment and data decrement messages as if they were Data Entry values

[x] Send Full NRPN (4 MIDI messages)

When checked, causes MIDI-OX to send both MSB and LSB of the NRPN value. When unchecked, only the MSB (Ctrl-6) is transmitted.

Sysex Mapping

You can map normal MIDI messages and NRPNs to System Exclusive strings. When you choose SysEx as the Output Event Type, the edit interface changes to allow entry of a SysEx string. Actually, any string may be entered here (for instance multiple normal MIDI messages), but any string will be sent via the low-level Windows **midiOutLongMessage()** API function. This function call sends a buffer instead of a single MIDI message. MIDI-OX SysEx string mappings are limited to 1022 MIDI bytes or less. They should be entered as hexadecimal bytes (example: "F0 7F 00 02 01 F7").

There are three special **substitution** characters you can use within SysEx strings: **FC**, **FA**, and **FB**. In a mapping, they are replaced by values from the MIDI message which triggers SysEx. They each result in 8 bits of data (a byte). They have the following meanings.

Code	Replaced by	Notes
FC	Channel	Values: 0 - 15, taken from the MIDI Message Status Byte
FA	Data1	Values: 0 - 127, The First MIDI Message Data Byte
FB	Data2	Values: 0 - 127, The Second MIDI Message Data Byte

Example of SysEx Substitution:

SysEx String:	F0 00 00 13 FC FA 10 00 FC 00 FB 00 F7
MIDI Message:	92 3F 48
Result:	F0 00 00 13 02 3F 10 00 02 00 48 00 F7

Map Example 1 (Keyboard Split)

We want to split the synthesizer keyboard into two instruments so that when we play it, all notes below G3 (MIDI note #55) are played by a bass instrument, while all those notes G3 and above are played by an organ. We will assume the synthesizer is playing back the organ patch on channel 1 and the bass patch is assigned to Channel 2. We'll map all notes below G3 onto channel 2, while all the others are just passed through on channel 1.

The first input filter finds all the Note On events on channel 1 which fall in the range from MIDI note 0 to 54. Notes that meet this criteria are simply mapped to channel 2; in fact that is the only parameter which is changed here:

Figure 1 - Keyboard Split

Input						Output					
Chan	Message	Value 1		Value 2		Chan	Message	Value 1		Value 2	
		Min	Max	Min	Max			Min	Max	Min	Max
1	NoteOn	0	54	x	x	2	x	x	x	x	x
1	NoteOff	0	54	x	x	2	x	x	x	x	x

Translating this mapping into English would read something like: on channel 1, any **note on** messages for notes 0 to 54 (with any velocity) should be changed to channel 2 without changing any of the other parameters. In a similar way, all note off messages in the same range are also mapped to channel 2. If we didn't do this we might end up with stuck notes (if our keyboard sends true note off events).

Map Example 2 (Reverse Keyboard)

This mapping accepts **any note on** message on channel 1, and interpolates the note number from the input range into the output range. Notice that the ranges are exactly reversed so that playing the lowest note on your keyboard sounds the highest pitch (and vice versa). After inverting the note number, it sends the message, still on channel 1, out the port.

Figure 2 - Reverse Keyboard

Input						Output					
		Value 1		Value 2				Value 1		Value 2	
Chan	Message	Min	Max	Min	Max	Chan	Message	Min	Max	Min	Max
x	NoteOn	0	127	x	x	x	x	127	0	x	x
x	NoteOff	0	127	x	x	x	x	127	0	x	x

This fact also points out another **feature**: mapping is very powerful; so much so, that it allows you to do things which may not make much sense, musically or otherwise. **Caveat emptor...**

Define Mapping Dialog

This window has the controls you need to define or edit your maps. You get to this window from the main Translation Mapping Dialog by selecting Insert or Edit, or by double-clicking a highlighted map entry.

In the Input section, the word **Any** in the Channel or Event Type boxes means exactly that; if they both say Any, then any MIDI event on any channel will be considered for mapping. A "-1" in the four Value boxes has the same meaning, i.e. all values will be mapped. These special cases will be represented by asterisks when you return to the Map Dialog. If neither Any nor -1 appear, then only events that match the actual values entered will be mapped.

In the Output section, the words **Match Input** in the Channel or Event Type boxes means that whatever channel or event is selected in the input will go out on the same channel or as the same type of event. Values of "-1" in the Value boxes have the same meaning; the values will pass through unmodified. Again, these will show as asterisks in the Map Dialog. When actual values appear in any of these boxes, it means that the corresponding values in the Input section will be mapped to these new values.

You can cause the input Value1 to be mapped to output Value2, and input Value2 mapped to output Value1. If only one of the check-boxes is selected ("*Use input value 2*" for instance), then Value2 would be mapped to both output Value1 and Value2. This data swapping can be particularly useful for mapping 2 byte MIDI messages to and from 3 byte MIDI messages.

You can specify that the original message be duplicated and subjected to further mapping or output by checking the **Pass original value on (Clone)** box. If all mappings have this value enabled, the original message is guaranteed to be output, regardless of any mapping. You can use this feature to create multi-message mappings: outputting NRPns or chords for a single input event.

Map Example 3 (Patch Mapping)

Suppose you wanted your lead instrument on channel 1, bass on channel 2, pads on channel 3, and drums on channel 10 and you wanted a quick way to switch patches appropriate to this setup. You could assign the bottom end of the keyboard to send pre-selected patch changes depending on the channel you were currently playing on. To do this you could set up a map for channel 1 that mapped a Note On of Value 0 to a patch change on the same channel of Value (whatever patch number you wanted as the lead instrument). A Note On of Value 1 could be mapped to the next lead patch number, etc. If your keyboard doesn't go down to 0, just set the Input Value number to the lowest note number it does send. Or the highest value. Whatever works best for you. You would do the same scenario for all the channels you want to map this way. If your synth accepts patch changes on channel 10 to get different drum setups, then you could do the same thing on channel 10. The included Patch Mapping.oxm sample map demonstrates this technique.

If you turn on the computer keyboard, this can be a fast way to send patches using the Patch Mapping technique. Hitting F2 with the keyboard active puts it in the bottom range, so the Z key sends note number 0 on the current channel the keyboard is set to use (change channels with the key pad Plus (+) and Minus (-) keys). If you are also using the computer keyboard to play from, then hit F6 (or whatever octave you want to play in) and you can instantly hear the new patch. See the [Keyboard](#) section for more information about using it.


MIDI Yoke Junction

MIDI YOKE is a MIDI Patch Cable driver. It is a Windows (Versions 3.1, 95, 98, Me and NT, 2000) multimedia driver.

MIDI YOKE is used to connect any Windows MIDI Application outputs to any other Application's inputs. The MIDI data stream is passed directly from output to input. This allows you to connect the MIDI output from one program to the MIDI input of a different program. MIDI Yoke can be configured to provide a varying number of MIDI Ports (from 1 to 8). In addition, each port allows multiple opens of both input and outputs: up to 3 openings per port. This flexibility provides for almost any configuration imaginable.


This release of MIDI Yoke adds a MIDI Yoke Beta driver for Windows NT. Although the Interface and functionality of the NT version is very similar to the Windows 95/Windows 3.1 version, the underlying architecture is quite different. The multi-threaded nature of the NT driver seems to prevent detection of MIDI Feedback, but feedback does not lock up the machine either, as it can under Windows 95.

MIDI Yoke Installation

Under Windows 95 (98,Me), you install the MIDI Yoke Junction via the Control Panel '**Add New Hardware**' applet. Click here  to start the Add New Hardware wizard.

Answer '**No**' to searching for hardware. Choose **Sound, video and game controllers**.

Choose '**Have Disk**'. Browse to the directory containing the installation files (MIDIYOKE.DRV and OEMSETUP.INF). Press **OK**...

During installation a configuration dialog is presented. The dialog allows you to specify the number of MIDI Yoke ports that should be enabled, and the Feedback detection desired. You can also change the controller number used to detect feedback. You can access this dialog later on too, once the driver is installed, and the machine restarted. To do so, Click here  to open the Control Panel **Multimedia** applet, choose the **Advanced** tab, expand '**MIDI devices and Instruments**', double-click on any of the MIDI Yoke port connections, and press **Settings...** You can also **Remove** the driver using this applet if need be.

Warning There is a bug in Windows 95 that limits the total number of MIDI Ports to roughly 11. This total includes each MIDI Yoke port plus any other ports presented by other drivers, so set the number of MIDI Yoke ports such that the total number of MIDI ports is less than 11. If you exceed this amount it may prevent the driver from loading at bootup time. Another symptom of this is receiving an MMSYSTEM error when you attempt any Windows MIDI activity. This bug seems to be fixed for Windows 98.

MIDI Feedback

The powerful nature of MIDI Yoke requires that a bit of care be exercised in its use: do not connect the outputs of one port to the same number inputs within a single application. If you do, it will cause MIDI feedback (loopback) - this phenomena will bring a computer to it's knees, and likely crash the system. Version 1.50+ of MIDI Yoke attempts to detect MIDI feedback by any of 3 configurable methods.

The first method was supported in previous versions: it simply sends an undefined controller number out the input port every so often (once every 50 messages). It looks at the output port to see if the controller shows up: if it does it assumes MIDI feedback and disables the driver. If you are plagued by unexplained controller messages (default is 103 decimal - 76 hex), you have probably selected this method.

The second (new) method analyzes the rate of messages traveling through the driver. If they exceed a certain threshold (2048 messages per second by default), the driver assumes MIDI feedback and disables the driver.

The third (new) method combines the previous two: it analyzes the data rate. If the rate exceeds the threshold, the driver sends an undefined controller out the input. If the controller shows up at the output, MIDI feedback is detected, and the driver is disabled. To recover from any of these conditions, simply close all connections attached to both ends of the particular MIDI Yoke port. After that the port may be reopened (but common sense would dictate that you should configure the routing differently).

VENDOR INFORMATION

MIDI Yoke driver may not be re-distributed without obtaining prior written permission. Since MIDI Yoke is freeware, our only means of compensation will come from commercial distribution of it. We welcome any and all proposals from individuals or corporations that would like to commercially distribute MIDI Yoke along with their other products.

Contact

JamieOConnell@midiox.com

DISCLAIMER

MIDI YOKE JUNCTION DRIVER IS PROVIDED WITHOUT ANY WARRANTY, EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO FITNESS FOR A PARTICULAR PURPOSE.

MIDI Yoke Installation NT, Win 2000

Under Windows NT and Windows 2000, use the Control Panel Multimedia Applet to install MIDI Yoke NT. Select the Advanced tab, highlight the **MIDI Devices and Instruments** section, and press **Add**. Choose **Unlisted or Updated driver**, and browse to the directory containing the drivers (**myokent.dll** and **oemsetup.inf**). Select the **MIDI Yoke NT** driver from the list. Make sure you don't leave an expanded **MYOKENT.DLL** in your MIDI-OX directory. If Windows NT finds this file it attempts to load it instead of the system one, and anomalies result. It's perfectly okay to leave the compressed version in your MIDI-OX directory (**MYOKENT.DL_**)

Options

There are a large number of options you can use to configure MIDI OX exactly the way you want it.

[MIDI Filter](#)

[Data Mapping](#)

[MIDI Devices](#)

[General](#)

[Select MIDI Player](#)

[Customize Port Names](#)

[Data Display](#)

[File Associations](#)

[Configure Status](#)

[Monitor Font](#)

[Colors](#)

[Configure Buffers](#)

[Pass Sysex](#)

[Customize Toolbar](#)



Pass Sysex

You can choose whether or not you want MIDI OX to monitor Sysex messages. If you are monitoring Sysex messages, then this menu command will be checked and the toolbar icon will show cables connecting the PC to the MIDI device. If you do not monitor Sysex messages, then any sysex messages that appear at the input ports will not pass through to the output port.

Note You must have Pass Sysex enabled to use the Sysex Window.

Middle C Octave

You can choose what Octave you prefer to have considered Middle C. This will map MIDI note number 60 to your choice. This will only affect the text displayed on the monitor; it does not alter the MIDI data stream.

Data Display Format

You can choose whether or not to display the number format in the monitor as decimal or hexadecimal.

Monitor Font

You can select the font you want to use in the monitor window. The font must be a fixed font so that the column spacing will stay intact. This means the only fonts you will have to choose from are fixed fonts, not all the fonts installed in your system.

Monitor Colors

You can select the colors you want to use for the text and the background in the monitor window. These choices will be remembered from session to session.

General Options

Tooltips

Display Patch Names

Middle C Octave

Running Status

Configure Status

Brings up a dialog that allows you to select what items are shown in the Status window.

Display Patch Names

If you choose this option the instrument name for a particular patch number will be shown in the monitor.

Configure Buffers

The buffers referred to are the Sysex buffers. The preferred place to configure the sysex buffers is from the Sysex View - see the [Configuration](#) topic. However, you can set the size and number of buffers from here as well.

Customize Port Names

If you have more than a couple of MIDI devices, it can be very confusing to remember which device is attached to which port. In an attempt to alleviate this situation, we have introduced the concept of **Port Names**. By customizing a Port name, you can specify how it should be displayed elsewhere in MIDI-OX. For example, instead of **SB 16 External Out**, you could name it **SB Out: Yamaha MU80**. Everywhere a port is referred to (MIDI Port Activity View, MIDI Devices dialog) it will use the customized name. When you press the Display devices button, or choose View | Device, it will display both the new and the original name. You can also see both names in the Options | Customize Port Names dialog.

Editing Names

To edit a Port name, click in the left (Custom Name) column to select the name. Then click again to invoke the edit control (or press **F2**). Type in the new name; the original name is always displayed in the right column. The Port icon color represents whether this is an input or output port. Input ports are colored Green; outputs are Light Blue. To remove a custom name, edit it and delete the text. It will be replaced by the original hardware port name.

After you have edited the names to your liking, press **OK** to save your changes; choosing **Cancel** will discard changes. The Custom Port names are saved globally. All running instances of MIDI-OX share the same custom Port names.

Select MIDI Player

This option allows you to select an external MIDI player. We install a small player called [MidiBar](#) and select this player by default. However, here you can configure any external program to run when you select the [Actions | Play MIDI](#) command.

File Associations

MIDI-OX allows the addition and removal of file associations with itself and MidiBar. After a file type is associated, double-clicking on it in Windows Explorer performs an action. Right-clicking on an associated file in Explorer offers a menu of action choices. MIDI-OX uses the following rules:

* If no instance of MIDI-OX is running then one is launched and the action is performed.

* If any instances are already running then the action is performed by the first instance located and a new instance is not launched.

When you choose **File Associations** from the **Options** menu a dialog appears with the three file types that MIDI-OX can associate. Existing associations are indicated by a checkmark. To add an association add a check mark to the desired type. Remove associations by clearing the check mark.

Map Files

This file type (*.oxm) stores the definition of a set of message mappings. See [Data Mapping](#) for more information about map files. The action performed on this file type by MIDI-OX is to load the specified map into an instance of MIDI-OX.

SysEx Files

This file type (*.syx) stores one or more system exclusive messages. The association includes two possible actions: **open** (default) and **send**. The default action is to load the specified sysex file into the SysEx Command view. From there you can edit and/or send the file. The **send** action sends the file immediately. You can right-click on a SysEx file in Explorer and choose the **send** action from the menu.

MIDI Files

This file type (*.mid) encapsulates a MIDI composition. The association is made to the MidiBar applet rather than to MIDI-OX itself. There are two possible actions: **open** (default) and **play**. The default action is start MidiBar and load the file ready for playing. You can also right-click a MIDI file in Explorer and choose **play** and MidiBar will start playback as soon as the file is opened.

Also see

[MIDI-OX Command Line](#)

for additional ways to automate MIDI-OX with certain file types.

MIDI-OX Command Line

MIDI-OX can react to file names on the command line. The actions are similar to those available through file associations. An additional option for command line specification is to place a profile file (See [Profiles](#)) on the command line. Whenever a profile is found on the command line a new instance is launched and that profile is loaded. Whenever one of the other file types is specified, an existing instance is searched for and used if found.

Usage

```
MIDI-OX "profile.ini" [/open | /send] "sysexfile.syx" [/open] "mapfile.oxm"
```

Usage Notes: The word, **MIDI-OX**, represents the full path to the **midiox.exe** file; "**profile.ini**", "**sysexfile.syx**", and "**mapfile.oxm**" represent the full path name of the corresponding file type.

Notes and Restrictions

Quotes are required around file names with spaces in them.

You can only specify one file of any particular type. You can't have two SysEx files on the command line, for instance, but you can have a Profile, a SysEx and a Map file.

MIDI-OX determines what type of file is on the command line by analyzing the extension for Maps and SysEx. Thus SysEx files must end in **.syx** and map files must end in **.oxm**. A profile must contain all ASCII text.

Mapping has to be turned on for a map to take effect. Specifying a map file on the command line only loads the map. If the previous map was turned on, the newly loaded one will be too.

Example usage

You might drag **midiox.exe** from Explorer on to the desktop or right-click on the desktop and select new shortcut and specify MIDI-OX to place a shortcut on the desktop. If you right-click the shortcut and select Properties you get a window where you can specify the Target. The MIDI-OX path will already be in the Target box. Place the cursor in the Target box at the end of **midiox.exe**, add a space and then type in your command. For example:

```
MIDI-OX "c:\Program Files\Midiox\my profile.ini"
```

Or

```
MIDI-OX "c:\Program Files\Midiox\my profile.ini" /send c:\SysExDir\MySysEx.syx
```

Or

```
MIDI-OX /open c:\Midiox\Map\MyMap.oxm /send "c:\My SysEx Dir\My SysEx.syx"
```

MidiBar

MidiBar is a lightweight MIDI player that uses the MCI services provided by Windows. There are known bugs with the Windows-provided sequencer that lead to problems when songs have large amounts of data or large amounts of sysex. However, it handles "normal" songs just fine. MidiBar is most useful if you choose a virtual port for its output and the same port as input to MIDI-OX.

MidiBar will accept the name of a MIDI file on its command line so you can set up shortcuts to favorite files or make MidiBar your default MIDI Player if you want. You can also associate MIDI files with MidiBar. See [File Associations](#). There are keyboard shortcuts for commands if you prefer them over using the mouse.

There are additional options available by clicking on the MidiBar icon in the upper left corner to get the system menu. Here you can specify if you want MidiBar to minimize to the system tray or to the task bar. You can choose if you want it to always be on top of other windows. You can choose if you want MidiBar to be a single instance. If you choose this option then files passed in on the command line or via activation by double-clicking them in Explorer (if you have MidiBar associated with MIDI files) will use an instance already running. Otherwise another copy of MidiBar will start up.

Click any part of the picture below for a description of each part.



Select the port that MidiBar uses. Keyboard shortcut "m".

Standard File Open dialog. You can also drag and drop one or more files from Explorer. Keyboard shortcut "o".

Start Playing the MIDI File. Keyboard shortcut "p".

Pause Playback. Keyboard shortcut "I".

Stop Playback. Keyboard shortcut "]".

Moves back in the file by 10 seconds. If held down, continues to move back in 10 second intervals. Keyboard shortcut "<".

Moves ahead in the file by 10 seconds. If held down, continues to move ahead in 10-second intervals. Keyboard shortcut ">".

Shows elapsed time

Shows total length of song

The name of the song appears in the title bar.

Status Bar

The status bar is displayed at the bottom of the MIDI OX window. It describes actions of menu items as you navigate through menus. When it is not displaying these messages, the left area shows the name of the selected MIDI output port. The next pane shows the number of input ports selected.

The right areas of the status bar indicate the following situations:

Indicator	Description
LOG	Logging is enabled.
REC	Recording is enabled.
SYX	Sysex is enabled.
KYB	The Keyboard is enabled.
MAP	MIDI Data Mapping is enabled.



Status Window

This window shows the most recent status of various MIDI messages on all channels. It will update as messages are received, even while closed. See [Configuring the Status Window](#) under the Options menu to determine what information you would like to see



System Exclusive Window

Note: You must have **Pass Sysex** enabled from the **Options Menu** to use the Sysex Window. If Sysex is enabled the letters **SYX** appear in the status bar at the bottom of the main MIDI-OX window.

You can open this window by selecting **Sysex** from the View Menu, or by selecting the icon on the [toolbar](#). Commands are available from the Menu or by right-clicking over one of the two windows. In the latter case, a small menu pops up with commands appropriate for the situation. To close the Sysex View select Quit.

The following Help topics exist for this window:

[Overall Concept](#)

[Loading Files](#)

[Sending Sysex Data](#)

[Sending a Sysex File](#)

[Receiving Sysex Data](#)

[Editing](#)

[Saving](#)

[Printing](#)

[Find and Find/Replace](#)

[Appearance Options](#)

[Configuration](#)

[MIDI Scratchpad](#)

[Compare Windows](#)

Overall Concept

Note: You must have **Pass Sysex** enabled from the **Options Menu** to use the Sysex Window. If Sysex is enabled the letters **SYX** appear in the status bar at the bottom of the main MIDI-OX window.

This window allows you to communicate with your MIDI gear via system exclusive messages. The exact form of these messages is dependent on the gear and should be outlined in your manual.

The view has two windows labeled Command Window and Display Window. The general idea is that you would type in or load from a file a sysex command string and select **Send** from the Command Window menu. If you expect the command to return a sysex message, select **Send/Receive** from the Command window menu. If you are receiving, a window will pop up showing the number of bytes that are being received. After you press OK, the bytes themselves will appear in the Display Window.

In addition, these windows can be used to send MIDI data other than Sysex data by typing, or loading from a file, lines of MIDI data that you want to send and selecting **Send Scratchpad**. See [MIDI Scratchpad](#) for more details.

The Load Command

Note: You must have **Pass Sysex** enabled from the **Options Menu** to use the Sysex Window. If Sysex is enabled the letters **SYX** appear in the status bar at the bottom of the main MIDI-OX window.

Selecting Load from the Command Window menu brings up a standard file open dialog box. In the List box labeled "Files of Type" you can choose the type of file you want to load.

The most common usage would be to load a text file that you previously created containing one or more sysex commands. These would represent in plain text the bytes that will make your particular equipment respond to a command. Although they are commonly referred to as "dump request macros" (DRMs), meaning that they "request" certain specific information and the synth responds by "dumping" the information back to the requesting unit, they can also be short (usually) commands that just change a particular setting without requesting any kind of return information.

The text files that the file open box will look for by default have extensions that end in ".txt", ".ini", or ".mox".

If you choose a binary file type, such as the Sysex type, the bytes will be translated into "human readable" form, such as "F0 18 04 ", etc., a form that we call the "Plain Hex" view. Each byte in the file is shown as its hexadecimal value, with each byte separated by a space. This is probably the same type of format that the manual for your MIDI instrument uses to explain the various types of sysex messages.

If you choose Formatted Sysex (or other binary type), then the bytes will be translated into a standard type of hex view, such as:

```
000010 00 74 00 20 00 47 00 72 00 61 00 6E 00 64 00 7F .t.G.r.a.n.d..
```

where the first six numbers show the position in the file of the first byte in the line (in hex), followed by 16 hex numbers representing the actual bytes, followed by an ASCII translation of those bytes, if any. If the bytes do not represent an ASCII letter, number or symbol, they are represented as a period. In the example above, the first byte (which is the 16th byte in the file as shown by the "000010") is "00", which has no ASCII printable value. The second byte is "74", which translates to an ASCII value of "T". The fourth byte in the line has a hex value of 0x20, which is ASCII for a "space", and so on.

Sending Sysex Data

Note: You must have **Pass Sysex** enabled from the **Options Menu** to use the Sysex Window. If Sysex is enabled the letters **SYX** appear in the status bar at the bottom of the main MIDI-OX window.

You can send data from either window. If you select data in either window (highlight it by selecting the text with the mouse or keyboard commands), **only that selection will be sent**. All of the data, or just the selected data, will go out as bytes. We only check that the data is a valid hexadecimal number before we translate it into a byte. Thus you can send any number of things that your equipment does not understand.

The Send/Receive Command causes the view to go into "receive" mode after the data is sent.

One possible way to use the view would be to load (or type) a command that requests a configuration message, say a request for the overall volume. The configuration is returned and showed in the display window. Usually your equipment will format this returned configuration in a way suitable to be returned. You might want to change the bytes that represent the value, say to lower the volume from 127 (0xF7) to 100 (0x64). You could do this in the display window and then choose Send from the Display Window menu to send the new volume specification to your gear.

Another useful way to use the two windows is to load a text file in the Command Window that has all possible messages you might want to send, perhaps along with text that explains the usage of each command. Then you can select (highlight) only the particular command you want and then choose Send or Send/Receive.



Sending a Sysex File

You can choose to send sysex files without previewing them in the Command Window by choosing Send from the file menu.

Receiving Sysex Data

Note: You must have **Pass Sysex** enabled from the **Options Menu** to use the Sysex Window. If Sysex is enabled the letters **SYX** appear in the status bar at the bottom of the main MIDI-OX window.

The three possible ways to begin receiving Sysex are:

- 1 Load (or type) a command in the Command Window that will cause a response from your gear and choose Send/Receive Sysex.
- 2 Choose Send/Receive Sysex from the Display window menu, providing there is something in the Display window to send.
- 3 Select Receive Manual Dump... from the Sysex menu item.

In Manual Dump mode, all sysex that is received is diverted into the Display Window until you select OK from the Receive dialog box.

See [Configuring](#) for information about appending bytes to the receive buffer.

Editing Sysex Data

Note that only text files or binary files opened or displayed as Plain Hex can be edited. Files displayed as Formatted Hex cannot currently be edited without swapping the type of view to Plain Hex. This may change in the future, but in an attempt to alleviate any distress this may cause, the selected byte or the byte where the cursor is will track from format to format. In other words, a byte selected in one format will appear selected when you swap formats. In addition, if you select or put the cursor in front of an ASCII translation in the Formatted view, the particular byte represented will be highlighted when you switch to the Plain format.

An editable format can be altered in the standard ways; for example a line of text can be selected and copied (CTL + C) or cut to the clipboard (CTL + X) and inserted at the cursor (CTL + V). There is quite a bit of freedom in this approach, and this requires a corresponding attentiveness to detail in the user.

If you choose to save the file as a binary file, for example, the success of this venture depends entirely on you. What we rely on is that two successive characters represent a valid byte, for example "F0". If the "F" is followed by "G" or a space or some other invalid character, then the F is thrown away and we search for the next valid **pair** of characters.

Valid hex characters are any number and the letters a to f or A to F. For the letters, upper and lower case are equivalent. So "F" is equivalent to "f" and it makes no difference if you mix upper and lower case.

It is not necessary for the bytes to be separated by spaces, so "F01804F7" is equivalent to "F0 18 04 F7" and also equivalent to "F0x18x04xF7", because the invalid character "x" is just disregarded. But do note that if the separating character is a valid hex character then the results will not be what you expect. "F0f18" would be interpreted as "F0 F1" and the 8 would be thrown away.

Saving Sysex Data

You can save the contents of either window with the appropriate menu commands. These work in the standard way with the following unique aspect. The Save dialog box offers the option of saving the contents as binary data **or** as a plain text file. Click the down arrow in the "Save as Type" box for a list of choices.

With the "plain text" option, you save what you see. If the control has "F0 18 04" in it, then that is exactly what is saved. You could open the file in Notepad and see "F0 18 04". File manager would report it as a file that was 8 bytes long, one byte representing "F", the next "0", etc... For your convenience, the file open window is keyed to look for the little-used extension ".mox", so you might find it convenient to use this extension. At the same time it looks for ".mox" files, it also looks for ".txt" or ".ini" files, so these would also be good choices.

If you save it as a binary file, then it would be three bytes long. The "F0" would be translated into 1 byte, as would the "18" and the "04". If you loaded it into notepad you would see three black squares, meaning that notepad can't show the letter those bytes stand for, because they don't stand for any letters. The standard and recommended file extension for data in "raw" sysex format is ".syx".

MIDI-OX can deal with sysex that is presented to it in either form. If the format is a textual representation of the bytes, then that will be converted into a binary format before being sent out as sysex data.

In most cases, you would want to save any "dumped" data as binary data, however. This is both the most efficient and guarantees that the data can be loaded into some other software, such as a sequencer.

As a further note, the above example would not be a valid sysex message for any conforming piece of MIDI gear, as it does not end in F7. What we assume is that you know exactly what you want to do and we don't impose any kind of "rules". Similarly, we don't impose any error checking on the byte values, except that they must be valid hexadecimal numbers. Thus, "FF" is a valid hexadecimal number and would be saved as a byte, even though that is an "illegal" value according to the current MIDI specification. "FG", for example, is not a valid hex number. If you typed "FG" then that value would be skipped and no attempt made to translate that into a byte.

Printing

The contents of either window can be sent to a printer and some rudimentary options regarding the appearance of the page can be altered via Page Setup. The default printing margins are 1 inch. If you make any changes to this then they will be remembered.

Find and Find/Replace

Find and Find/Replace are implemented in the standard fashion, allowing you to search for specific bytes or phrases. Note that just as editing is not supported in the Formatted view, neither is Replace supported in this view. However, you can use Find in the Formatted view and then switch view types to do editing.

Appearance Options

Each window can have a different color scheme for the text and background. Each window can have word wrap turned off or on. If a window is showing either **Plain Hex** or **Formatted Hex**, then the format can be swapped to the other type. Note that if the Command Window has been loaded with a text file, then choosing to swap the format will have no effect. You can right click in either window and get an appropriate menu for that window.

Configuring Sysex Options

This menu item brings up a dialog box giving you the possibility to alter various settings:

- 1) You can change the **size and number of the 'low-level' sysex buffers**. The Input buffers (the ones receiving data) and the Output buffers (the ones sending data) can be configured independently. See [More About Buffers](#) for a more extensive discussion. We recommend 32 to 64 buffers of 512 bytes as an all purpose solution that works well on a variety of machines.
- 2) You can choose to have **Save Warnings** turned off. By default, any time you start a dump and you have already received something that you haven't saved, you will be asked if you want to save it. If you want to take on this responsibility for yourself, then check this option and the Save prompts won't occur.
- 3) When sending Sysex, some gear needs a break after the end of a SysEx message (signaled by F7) to process that message before it can handle another message. Since some messages, like loading an entire bank, frequently are composed of smaller messages, your gear might not be able to handle a composite message at full speed. By checking the **Delay After F7** box and specifying a time in milliseconds, any composite messages will wait the specified delay before sending the next message. Or, if you just want to break a large composite SysEx message into individual chunks and don't need the delay, you can check the box but specify a delay of 0.
- 4) You can also optionally specify a **delay after each buffer** is sent. Some gear might have a small buffer to receive SysEx (say 256 bytes) and might not be able to process this data fast enough, so the bytes after the first 256 are ignored. In this case, you could set an output buffer size of 256 and then specify a delay of 100 milliseconds (or whatever is necessary). This delay will allow your gear enough time to process the buffer before the next 256 bytes are sent. You should be able to tailor the speed to fit any situation. As an extreme example, you can select one buffer, one byte in size and specify a delay of 1000. This means you would be sending data at the astounding rate of 1 byte per second!
- 5) You can choose to **append bytes** to ones you have already received. Suppose you wanted to save five altered patches for a particular song all in one sysex file that you could load every time you wanted to play the song. After choosing to append, you could send the five individual patch requests one after the other and the five patches would then be stored sequentially for you to save as one sysex file.
- 6) If you check **Fill Display Window as Bytes Come In**, the receive window will do what it says, fill up with data as it arrives. Since this consumes a certain amount of time, depending on the speed of your computer, you may wish to **not** fill the display window until all the bytes have arrived. This way all the time is devoted to retrieving data. The trade-off is that all the data will consume memory as it fills up waiting to be displayed.
- 7) If **Show F7 in Red** is selected then all the F7's will be colored red. It makes it easier to look at a bulk dump and see where the individual messages are. On slower machines or huge bulk dumps, going through and finding all the F7's can take a certain amount of time. You might want to not have this checked if you know you will be dumping a huge amount of data.
- 8) If **Save Dump Directly To a File** is checked then the data will not appear in the sysex window at all but will just be written to a file of a name you specify. Once again, this is for huge bulk dumps which you probably don't need to look at anyway. It is also a speedier operation and uses very little memory.

More About Buffers

The entire multimedia system under Win9x is still 16 bit. This is why there is a 16 bit DLL to handle the low level conversation with the MIDI driver. (Win NT works in a similar way, but the DLL is 32 bit.) The low level buffers are allocated and maintained by this 16 bit DLL. They are set up in a circular arrangement, meaning that as they are filled by the MIDI driver on the 16 bit side, they are emptied by the application on the 32 bit side, and returned to the MIDI driver to be filled again. The default is for 16 buffers that are 256 bytes in size.

Imagine a train with 16 small cars on a circular track. The cars are loaded in 16 bit land and emptied out in 32 bit land. By the time the last car is emptied in 32 bit land, the first car is back at the 16 bit station ready to be loaded again. The reason the size and number of the "cars" is adjustable is due to variations in the way individual MIDI drivers work. Some may work better with a few large buffers, but in general they usually work best with a larger number of smaller buffers.

One reason this is so is because most drivers will fill a buffer "car" until they come to an F7. If there are more bytes coming, the driver will then switch to the next buffer. As an example of how this works, an EMU Proteus stores its patches in banks of 64, with each individual patch being 265 bytes long and terminating with an F7. If you dump a bank of patches, the first buffer will be filled with the first patch, and then the driver will switch to the second buffer for the next one, and so on. For this scenario, you can see that the most efficient use of the low level buffers would be to have 64 of them, each 265 bytes in size. If you set the buffers larger, the extra space won't be used, and if you reduce the number, then some will have to be used more than once (although this wouldn't pose a problem). Thus, even though the entire bank is 16,960 bytes, you can see that one buffer of 16,960 bytes would be extremely inefficient, and probably won't even work. The one buffer would only get 265 bytes and would have to chuff around as quickly as it could to get the next 265 and so on, 64 times.

Each open input device (port) and each open output receives a full complement of buffers. Buffers must be locked and fixed in low DOS memory for the 16 bit drivers, as they are accessed at interrupt time. Because of this, if you have very many input and output devices that you wish to keep open, you probably will want to reduce the number of buffers handed to each port to an absolute minimum. If low memory runs out, you may not be able to launch any more programs unless you reboot.

MIDI Scratchpad

Either Command or Display windows can be used to send non-sysex messages in "raw" hex format. For example, a Middle C note-on message on channel 1 with a velocity of 127 would be:

```
90 3C 7F
```

If you type this line into one of the windows and select Send Scratchpad, the note should sound on the instrument selected as the MIDI out port (and not stop sounding until you send a message to turn it off, by the way). You can type in multiple messages, with each message on a new line.

```
90 3C 7F
```

```
90 3C 00
```

If you select a line, or multiple lines, then only the selected lines will be sent. Otherwise, all of the lines in the window will be sent. You don't need to separate the bytes with spaces and case is not important. So:

```
903c7f
```

works fine.

Compare Windows

When you choose this command, MIDI-OX compares the **data** in the Command Window to the data in the Display Window, reports the results, and puts an underline under any bytes that are different. Typically you might use this feature when you want to compare a Sysex dump you have received in the Display Window against a known-good dump that you can load in the Command Window.

System Menu

Besides the standard menu items the system menu has options to keep MIDI OX on top of all other windows, a toggle for releasing and acquiring the selected MIDI ports, and an option to minimize MIDI OX to the system tray.



On Top

If you select this command, MIDI OX will always appear on the screen on top of all other windows; a check mark will appear next to the command. Selecting this command again will clear the check mark and restore the normal Window behavior. The toolbar images change to reflect which state you are in, on top or normal.



Free MIDI

Choosing this command will release the MIDI ports while remembering which ones you have selected. If you have a MIDI driver that will only allow one client at a time, this will allow you to temporarily release it so that some other program can use it. After choosing this command, the menu item will change to read **Attach MIDI** and the toolbar button will change to show an empty bird cage. Pressing this button or choosing the Attach MIDI menu command allows you to re-acquire the MIDI ports that had previously been selected.



Minimize to Tray

Choosing this option causes MIDIOX to appear as an icon in the System Tray area of the taskbar rather than as a minimized program on the taskbar itself. When MIDI activity is detected, the icon will cycle through four different icons to reflect that activity. There are four icons included within the program, or you can supply your own custom icons. These custom icons should be labeled:

Tray0.ico

Tray1.ico

Tray2.ico

Tray3.ico

























and they should be placed in the same directory as MIDIOX.

If MIDIOX detects these icons when it starts up, it will use them rather than its internal icons.

Toolbar

The toolbar is displayed across the top of the application window, below the menu bar. The toolbar provides quick mouse access to many tools used in MIDI OX.

Also see [tooltips](#) and [customizing](#).

Click	To
	Start or Stop Logging
	Send a Sysex File
	Open the Sysex Window
	Opens the Controller Window
	Opens the MIDI Status Window
	Turns on the Keyboard
	Opens the Filter Window
	Opens the Devices Window
	Shows the Current Devices
	Toggles and shows state of recording
	Clears Display Window
	Toggles On Top status
	Toggles Opening/Closing the MIDI ports
	Toggles Sysex Monitoring
	Opens Help Menu
	Opens Input Monitor
	Panic Stop
	Opens Data Mapping Window
	Opens MTC Transport Window
	Runs a MIDI Player
	Opens NRPN Calculator
	Opens Port Status Window
	Opens Instrument Panel
	Selects Minimize to System Tray Options

Tooltips

By default the toolbar will show short hints about the actions of a particular button when you rest the mouse pointer over one of the buttons. You can turn this behavior off by turning off the check-mark next to Tooltips on the [Options](#) menu.

Customize Toolbar

You can choose which toolbar buttons are displayed on the toolbar and the order of their appearance. You can choose Customize Toolbar on the Options menu or you can double click the left mouse button on an empty part of the toolbar to get a dialog that lets you do this.

View Menu

This menu item lets you select from four different windows or choose to display the selected MIDI devices.

Control Panel

Instrument Panel

NRPN Calculator

MTC Transport

Midi Sync Transport

Sysex

MIDI Status

Input Monitor

Port Status

Device

Running Status



Input Monitor

You can choose to monitor the raw input data before it is filtered or mapped. This will open a second monitor window so you can compare the raw and altered data.



MTC Transport

The MTC Transport was designed to be used for diagnostic purposes mostly, but can be used to drive any equipment that can respond to SMPTE (MTC) sync. It can also be used to determine that a device which sends SMPTE is operating correctly.

As a test, configure MIDIOX to send to MIDIYoke port 1. Configure a program that will respond to MTC, for example, Cakewalk, to listen to MIDIYoke port 1.

Open a song in Cakewalk. Click the Clock Source button (or Settings | Clock...) to set to SMPTE (MTC). Click the Time Format (or Settings | Time Format) and enter a SMPTE offset of about 3 (00:00:03:00) sec - this gives it enough time to acquire sync. Pick any of the 4 standard Sync modes (24, 25, 30 drop, 30 non-drop) in Cakewalk. Press Play, and Cakewalk will start waiting. In MIDIOX, open the MTC Window, set the same sync mode you selected in CW, set a 0 offset, and press play. CW should now respond to the Time being sent by MIDIOX - In fact MIDIOX is driving Cakewalk and supplying it the Clock source. If you press Stop in MIDIOX, Cakewalk will stop. You can press pause to prevent rewinding. Cakewalk requires that you press Play each time you want to acquire Sync. To start at a place other than the beginning, you can dial up an offset in MIDIOX.

The **resolution** field controls how tight the timing should be: lower resolutions yield tighter timing. On slower machines, however, a very low resolution might cause the system to become sluggish, as many resources are given up to prioritize the timing.

Cakewalk can't generate MTC, but you can drive one MIDIOX instance with another instance using MTC -- just open the MTC window in each instance (and connect them with MIDIYoke).



Midi Sync Transport

The MIDI Sync Transport allows display and control of another device (or application via MIDI Yoke) via MIDI Clock Synchronization. It also has an option to simultaneously control the MTC transport: sending both MIDI Sync and MTC.

As a test, configure MIDI-OX to send to MIDI Yoke port 1. Configure a program that will respond to MIDI Sync, for example, Cakewalk, to listen to MIDI Yoke port 1.

Open a MIDI song in Cakewalk PA9 or earlier. Click the Clock Source button (or Settings | Clock...) to set to MIDI Sync. In Cakewalk, Press Play, and Cakewalk will start waiting. In MIDI-OX, open the MIDI Sync Window, set the tempo as desired and press play. CW should now respond to the Tempo being sent by MIDI-OX - In fact MIDI-OX is driving Cakewalk and supplying it the Clock source. If you press Stop in MIDI-OX, Cakewalk will stop. You can press pause to prevent rewinding. Cakewalk requires that you press Play each time you want to acquire Sync. To start at a place other than the beginning, you can dial up an offset in MIDI-OX.

The readout portion of the MIDI Sync view displays the most recent tempo and position received: when the transport is playing, events are rerouted back to the view, so the display updates. The readout also updates when MIDI clock are received from external sources and can be used as a MIDI Clock Sync monitor.

The **Tempo** in the MIDI Sync window controls how often MIDI Clocks (F8h) are sent. The **Accuracy** control determines the resolution of the Windows timer: lower values place more of a strain on the CPU, but yield higher accuracy.

The **Timebase** and **Meter** fields control how the **Measure:Beat:Tick** values are displayed. You can cause the MIDI clocks to be sent all the time the MIDI Sync view is open by checking **[x] Send Clock**.

Certain MIDI Sync devices operate this way, and allow a receiver to pre-determine the tempo that will be in effect upon receiving a MIDI Start or Continue. The transport can cause the Song Position pointer to be incremented or decremented by a measure by clicking the **Backward** and **Forward** Buttons.

The **[x] Sync and Send MTC concurrently** checkbox allows the synchronization of MIDI clock and MTC. When it is checked, the MTC view will be loaded (with its transport disabled), and both items will be controlled by the MIDI sync transport. When play is started, both MTC and MIDI Sync will be generated, and both views will stop upon pressing Stop.



NRPN Calculator

Calculates the single NRPN value when you enter the MSB and LSB or cracks the NRPN into MSB and LSB



Instrument Panel

This window allows you to use Cakewalk .ins files to define favorite patches. See a [complete discussion](#) about the panel for more information.



Port Status

This window shows MIDI Activity for each port with LEDs for each channel.

Running Status

Running Status is an option provided in the MIDI Specification that allows MIDI drivers to **not** send a status byte with a message when the status of the message is the same as the previous message. It is also used when writing MIDI files to disk. This reduces both the message traffic and the file size. However within Windows, there is no advantage to using running status as the Windows message size is the same whether running status is used or not. MIDI-OX will track running status if the option is selected, but there is no advantage to the average user to do so. This option is provided for testing drivers and should not be selected unless you know why you are selecting it.

Under the View menu you can select to see printed out in the monitor the current Running Status option and last status byte.

Welcome

Welcome to **MIDI OX**, the world's greatest all-purpose MIDI Utility! For those with time on their hands, you can read a brief [history](#) of MIDI OX and how it came by it's name. We have also included a [general description](#) of the utility. The rest of you will want to get right to work.

[Installation](#)

[Getting Started](#)

History

MIDI OX started out as a sample program written by Microsoft in 1991 called Midimon. Midimon achieved a certain measure of fame over the years because of an error in its **.DEF** that prevented it from being built. Jamie published a corrected and slightly enhanced version of Midimon on CompuServe shortly thereafter. A few years later, Jerry joined with him to continue to develop and enhance the original program. When Windows 95 came out, we developed 32 bit and 16 bit versions of the program in parallel. Eventually, awash in #ifdefs, we decided to split off further development into a single Win32 program. At that time we distinguished between the two programs by calling them MIDIOX16 and MIDIOX32. Now that there really is only one program, we are just calling it MIDI OX. We changed the name from Midimon because there is virtually none of the original code left anymore. The only legacy is the original idea of monitoring MIDI data.

The "OX" part of the name came about for two reasons. First, in the world of 8.3 file names we wanted a name that included "MIDI" and "16" or "32", leaving just two letters to work with! Second, we do consider the program a beast of burden hauling the data, so it kind of fit. Besides, Jamie already had written MIDI Yoke, and its relationship to MIDI OX is similar to the yoke on a team of oxen, so there it is.

General Description

MIDI-OX is a Windows 9x / Windows NT program. It is a 32 bit program which will not operate under earlier versions of Windows. MIDI-OX 32 is a multi-purpose tool: it is both a diagnostic tool and a System Exclusive librarian. It can perform filtering and mapping of MIDI data streams. It displays incoming MIDI streams, and passes the data to a MIDI output driver or the MIDI Mapper.

MIDI-OX is copyrighted freeware: this means it can be freely used by individuals in non-commercial environments.

Installation

If you've downloaded the self extracting executable and are reading this help file, then you've already accomplished the basic installation. Note that this is a "by the book" installation and includes standard Windows "un-install" capabilities. It will appear in the Control Panel Add/Remove Programs list.

Following is a list of files included with MIDI-OX.

MIDIOX.EXE	The program
MMCBK16.DLL	16 bit library (WIN 95)
MMCBK32.DLL	32 bit library (WIN 95)
MMCBKNT.DLL	32 bit library (WIN NT)
MOXLIB.DLL	Stuff library
MOXREXX.DLL	REXX Support
MOXKART.DLL	COM Automation
MIDIBAR.EXE	Stand alone MIDI Player
MIDIYOKE.DRV	Windows Multimedia multiplexing driver
MYOKE.TSK	MIDI Yoke Task (Win 95 / Win 3.1)
MYOKENT.DL_	MIDI Yoke NT driver (Beta)
OEMSETUP.INF	Install file for MIDI YOKE (Win 95/NT)
MIDIOX.HLP	Windows Help File
MIDIOX.CNT	Contents file for the help file
README.RTF	Last Minute notes and usual ReadMe
*.OXM	Sample MIDI-OX mapping files (Map Directory)
*.SYX	Generic System Exclusive files (Syx Directory)
*.REX	Sample REXX Scripts (Rexx Directory)
*.VBS	Sample automation Scripts (WSH Directory)
*.INS	Sample Instrument definition files (Instr Directory)
TRAYALT.ZIP	Alternate Tray Icons

All of these files are placed in the same directory where you chose to install MIDI-OX, except where noted. If you choose to install the MIDI-YOKE driver, then Windows will place a copy of midiyoke.drv in the System directory. If you're running Windows NT install the MIDI Yoke NT beta drivers. In this case the MYOKENT.DL_ file will be expanded and placed into the NT System32 directory.

Make sure you don't leave an expanded MYOKENT.DLL in your MIDI-OX directory; if Windows NT finds this file it attempts to load it instead of the system one, and anomalies result. It's perfectly okay to leave the compressed version in your MIDI-OX directory (MYOKENT.DL_)

To read more about MIDI YOKE and see why you might want to install it see [MIDI YOKE Junction](#).

See [MIDI YOKE installation](#) or [MIDI YOKE installation for Windows NT](#) if you want to install this driver.


Getting Started

The first thing to do is connect MIDI-OX to your MIDI device(s). Activate the [MIDI Devices](#) dialog window to choose from a list of all the MIDI devices installed on your system. You can select one MIDI Output and any number of MIDI inputs. You are not required to specify any Inputs if you only want to work with the Output, nor do you have to specify an Output device to work only with Inputs. See [Making Connections](#) for some example configurations.

Note If both the Input and Output windows are empty, then Windows is reporting that you do not have any MIDI devices. If you actually do have MIDI equipment, then you will have to install or re-install the device drivers for that equipment. You will not be able to do anything useful with MIDI-OX without at least one MIDI device. See [Removing MIDI-OX](#) for instructions on removing it from your system.

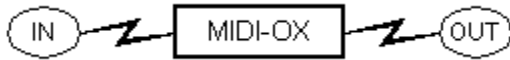
Once you have specified the devices to monitor and (optionally) send data to, you are ready to start. Any MIDI data appearing at a selected Input port will appear in the monitor window. You can use the [Control Panel](#) or the [Keyboard](#) to inject MIDI data. You can use the [Data Mapping](#) capability to alter the input data before it goes to the output stream. You can [Filter data](#) to specify that it does not appear in the output stream, or just to specify that it does not appear in the monitor window. You can watch both the [raw input](#) and the modified output if you wish. Any [System Exclusive messages](#) will also be passed through if you have selected this option. To actually work with System Exclusive data, see [System Exclusive](#). You can [log](#) any incoming data to a text file. You can see the [status](#) of the most recent MIDI messages and controllers.

Removing MIDI-OX

Click here  to activate the Add/Remove programs applet. Select MIDI-OX from the list and select the Add/Remove programs button.

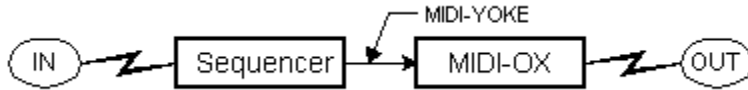
Making Connections

Figure 1



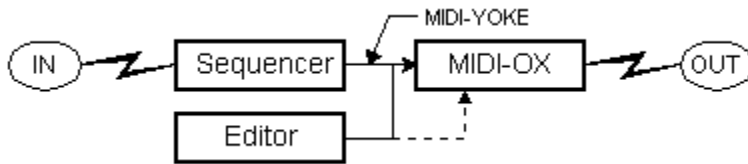
The MIDI Input port is routed through MIDI-OX to a MIDI Output port. This basic configuration allows you to test, monitor, and alter MIDI data sent from your external MIDI keyboard to your MIDI output port, either back to your keyboard or to external modules or a sound card.

Figure 2



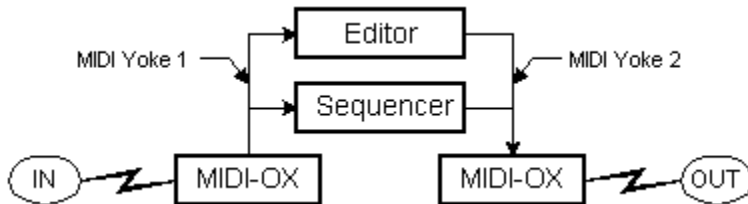
This figure shows your MIDI Input port connected to a software sequencing application. The Output port selected in the sequencer is one of the MIDI Yoke ports. The other end of the MIDI Yoke port is selected as an input device in MIDI-OX, and your MIDI Output port is selected as the Output device in MIDI-OX. Again, you can test, monitor, and alter MIDI data, this time as it comes out of your sequencer. Note that you could reverse the positions of MIDI-OX and the sequencer and you could interact with the data stream **before** it flows into the sequencer.

Figure 3



This shows the same configuration as Figure 2 with the addition of a patch editing or librarian application. Since a MIDI Yoke port can be opened by multiple clients, the editor is configured to use the same MIDI Yoke port as the sequencer. The dashed line shows an alternative configuration; the editor uses a second MIDI Yoke port and two inputs are opened in MIDI-OX.

Figure 4



You can run multiple instances of MIDI-OX; this configuration shows two copies running. The first MIDI-OX is connected to a MIDI Input port. Its output is connected to a MIDI Yoke port. The editor and sequencer have opened connections to that same MIDI Yoke port. Their outputs are connected to a different MIDI Yoke port, which is opened as an input to the second MIDI-OX. The second MIDI-OX is connected to a MIDI Output port. This configuration provides the opportunity to test, monitor, and alter MIDI data both as it enters and as it leaves the computer. In addition, both the editor and sequencer function as though they were connected directly to the MIDI input and output ports.

In conclusion, these example connections show only some of the possible ways to connect MIDI-OX and MIDI Yoke together with other applications and MIDI ports. There should be enough flexibility to connect your particular applications and equipment in whatever way you require.

Window Menu

Standard Windows Menu commands.

MIDI-OX REXX Interface

You can run REXX and Object-REXX programs from MIDI-OX. The programs can access MIDI data provided by a named REXX queue. The data can be processed in any fashion and then sent out the MIDI output port via MIDI-OX supplied functions.

Important MIDI-OX does not supply REXX, but will use the environment if available. Currently, only IBM Object-REXX is supported. If a REXX installation is not detected, the option is removed from the main menu, and the DLL is not loaded.

The REXX Dialog

Select a REXX program to run by typing in the path name in the Program box or push the button to select from an Explorer dialog. You can specify optional parameters in the Parameters edit line, or leave it blank. The parameters are supplied as a single command line string to the REXX program. You can specify a directory to start in by creating and/or adding to `moxrexx.ini`

[Options]

`REXXDir=C:\Program Files\Midiox\Script ;script directory`

Press the Launch button to run the program. The Trace screen displays any messages from REXX, particularly valuable when debugging. Note that the dialog can stay open or you can close it with the Close button without affecting the running REXX program. To stop running the program press the End Program button.

You can launch an editor to change the REXX program by pressing Edit. By default, this launches Notepad.exe. To specify a different editor, add to `moxrexx.ini`

[Options]

`editor=c:\lib\epsilon\epsilon.exe ; would use the Epsilon emacs editor`

There can be slight latency delays introduced when all input is diverted through the REXX program, but you don't have to divert input -- in fact it's not diverted by default -- it's normally sent to both REXX and directly to the output. This allows you to do echo delays and arpeggiator type programs in just a few lines of code. If you do want to divert input, check the Divert MIDI input box on the dialog.

The Queue

MIDI-OX uses a standard REXX queue construct. In fact, it is REXX itself that provides the queue. Data is entered into the queue by MIDI-OX in a FIFO (first-in, first-out) order, but the REXX program can remove the data in anyway it wants.

MIDI-OX will suggest a queue name that will be unique within the MIDI-OX instance. It is possible to override this name, even with the name of an existing queue in another MIDI-OX instance. The significance of the queue name is it creates a *private* queue, that *can* be shared among instances, if you don't rename it in the new instance. Since entries are de-queued by the REXX program when they are read, only one instance will read any particular message.

Real-time MIDI messages are entered into the queue as they are received, as long as the REXX program is running. By default, the data entered is the standard MIDI message: a character representation of the decimal data bytes. The representation can be changed by a function call `MOXSetQueueDataFormat()`, which is described later. Each queue entry message is three or four data values, possibly padded with zeros for MIDI messages that are shorter than three bytes. No running status is supplied or should be expected. A millisecond timestamp is also provided. This represents the time elapsed since the MIDI device was opened.

Examples:

15632 144 64 106 Note On at 15632 milliseconds: channel 1, note #64 (E4), velocity 106

2338 192 33 00 Program Change at 2338 milliseconds: channel 1, patch #33

The data can easily be parsed into separate variables and processed. For example:

PULL Status Data1 Data2

The above will move the next queue entry into the four variables named.

The standard REXX conversion functions can be used to convert between data formats. For instance you could split off the event and channel information from the status by:

Event = BitAnd(X2C(D2X(Status)), 'F0'x)

Chan = BitAnd(X2C(D2X(Status)), '0F'x)

If you always want the event and channel split off, you can have MIDI-OX do it for you by an alternate queue data format:

MOXSetQueueDataFormat("C")

...

PULL TimeStamp Event Chan Data1 Data2

The above will split off the event and channel information, and return the data in decimal format.

When MIDI-OX is requested to end a REXX program it will queue a special value, that alerts the REXX program to exit ('**END_DATA**'). If the program does not respond in a reasonable amount of time, you can forcibly terminate it. This prevents a stuck program from languishing.

MIDI Output

Once the data is processed, it can be returned to MIDI-OX for output. You use the **MOXOutputMIDI()** function to return the data.

Example:

```
ret = MOXOutputMidi( Status, Data1, Data2 )
```

Alternate functions exist to return the data channelized and/or in Hex:

```
ret = MOXOutputMidiC( Event, Chan, Data1, Data2 )
```

Expects a MIDI status in **Event** and the channel information in **Chan**. MIDI-OX combines it to form the output message.

MIDI-OX REXX Programs

The normal way a MIDI-OX REXX program operates is by polling the queue for data. If any is available, it is processed and output; if not, it should do any background tasks and then block while waiting for more input. A system semaphore is created with the same name as the queue and is pulsed each time MIDI data is queued.

Example:

```
/* REXX: Simple MIDI output loop */
call RxFuncAdd 'SysLoadFuncs', 'RexxUtil', 'SysLoadFuncs'
call SysLoadFuncs

qName = MOXGetQueueName()
oldq = RxQueue( 'Set', qName )

/* MIDI-OX creates a semaphore with same name as queue */
sem = SysOpenEventSem( qName )

if sem = 0 then
    running = 0
else
    running = 1

do while running
    do while Queued() <> 0
        pull timestamp status data1 data2
        if timestamp = 'END_DATA'
            then do
                running = 0
                leave
            end
        end
    /* ... do any processing. Then ... */
    ret = MOXOutputMIDI( status, data1, data2 )
    if ret <> 0
        then do
```

```

        call RxMessageBox( MOXErrorText( ret ), , 'STOP' )
    end
end

if running
    then do
        /* queue is empty: you could do other processing here */
        ret = SysWaitEventSem( sem )
        if ret <> 0 then
            running = 0
        end
    end
end

call SysDropFuncs
exit

```

If a parsing or runtime error occurs in the REXX program, the error or trace diagnostic is displayed in the **Actions | Run REXX...** dialog box. This information is invaluable for debugging REXX programs. Several REXX programs are installed during installation: these examples should be useful as a starting point for your own programs.

MIDI-OX REXX Functions

MIDI-OX REXX API functions are registered with REXX before a program begins, so they are always loaded and available to run in an efficient manner. The following routines are defined:

MOXGetQueueName()

Summary: Returns the name of the private data queue that MIDI-OX creates for REXX programs. This name is used for both the data queue, and for the name of a system semaphore that is used to wake up the REXX program when data is available.

MOXSetQueueDataFormat(format)

Summary: sets the format for the data entered into the queue. Calling this function will clear the queue of any existing data.

format: one of the following:

- 'X' output all values in Hexadecimal. Example: **status data1 data2 - 93 40 6F**
- 'C' split off the channel information from the event type. This causes an extra channel field to be inserted into the message. Example: **event chan data1 data2 - 144 3 64 55**
- 'S' Like 'C', but data is output in Hex: **90 03 40 4F**
- 'N' (Default) Queue values in decimal. Example **147 64 120**

Notes: This function is optional. It has the side effect of emptying (discarding) the queue each time it is called.

MOXGetSystemTime()

Summary: returns the result of the standard windows MME API, timeGetTime(), as a numeric text string. The value represents the number of milliseconds elapsed since the system was started.

MOXErrorText(err)

Summary: Returns an error message text string associated with the error number supplied. If an unknown error number is input 'Unknown Error - #err' results.

Err: a standard MIDI-OX REXX error number.

Number	Text	Description
100	Bad parameter: number required	Non-numeric input passed to routine.
101	Bad parameter: out of range	Value was out of legal range
102	Bad parameter: input required	Missing data
103	Bad parameter: unknown status	Illegal MIDI status
104	Bad Parameter: unexpected data	Parameter not allowed
105	Bad Parameter: unknown Argument	Unrecognized parameter
106	Error: out of memory	System Memory Error

MOXOutputMidi(Status, Data1, Data2)

Summary: Sends a MIDI message to MIDI-OX for output to the currently open port. All data is assumed to be in decimal format: an error will result if it's not.

Status: A combined event-channel message number. The standard MIDI spec. event type (Most significant 4 bits) and the zero based channel number (least significant 4 bits).

Data1: Depends on message type, but always a value between 0 and 127 inclusive. For a Note event the first data value is the MIDI note number; for a Program change it is the patch number.

Data2: Depends on message type, but always a value between 0 and 127 inclusive. For a Note event the second data value is the MIDI note number; for a Program change it is not used (and should be supplied as 0).

Return: This function returns 0 for success, or an error code for failure.

MOXOutputMidiC(Event, Chan, Data1, Data2)

Summary: Sends a MIDI message to MIDI-OX for output to the currently open port. All data is assumed to be in decimal format: an error will result if it's not.

Event: The standard MIDI spec. event type. Only the most significant 4 bits of this number will be set - the least significant bits will be 0.

Chan: The zero based channel number. This is a number between 0 and 15, that represents MIDI channels 1 through 16.

Data1: Depends on message type, but always a value between 0 and 127 inclusive. For a Note event the first data value is the MIDI note number; for a Program change it is the patch number.

Data2: Depends on message type, but always a value between 0 and 127 inclusive. For a Note event the second data value is the MIDI note number; for a Program change it is not used (and should be supplied as 0).

Return: This function returns 0 for success, or an error code for failure.

MOXOutputMidiX(Status, Data1, Data2)

Summary: Sends a MIDI message to MIDI-OX for output to the currently open port. All data is assumed to be in Hexadecimal format: an error will result if it's not.

Status: A combined event-channel message number. The standard MIDI spec. event type (Most significant 4 bits) and the zero based channel number (least significant 4 bits).

Data1: Depends on message type, but always a value between 00 and 7F inclusive. For a Note event the first data value is the MIDI note number; for a Program change it is the patch number.

Data2: Depends on message type, but always a value between 00 and 7F inclusive. For a Note event the second data value is the MIDI note number; for a Program change it is not used (and should be supplied as 0).

Return: This function returns 0 for success, or an error code for failure.

MOXOutputMidiS(Event, Chan, Data1, Data2)

Summary: Sends a MIDI message to MIDI-OX for output to the currently open port. All data is assumed to be in Hexadecimal format: an error will result if it's not.

Event: The standard MIDI spec. event type. Only the most significant 4 bits of this number will be set - the least significant bits will be 0.

Chan: The zero based channel number. This is a number between 00 and 0F, that represents MIDI channels 1 through 16.

Data1: Depends on message type, but always a value between 00 and 7F inclusive. For a Note event the first data value is the MIDI note number; for a Program change it is the patch number.

Data2: Depends on message type, but always a value between 00 and 7F inclusive. For a Note event the second data value is the MIDI note number; for a Program change it is not used (and should be supplied as 00).

Return: This function returns 0 for success, or an error code for failure.

MOXOutputSysEx(Data)

Summary: Sends a MIDI System exclusive message to MIDI-OX for output to the currently open port. All data is assumed to be in Hexadecimal format: Hex bytes can be separated by spaces. Example: F0 41 10 42 12 40 00 7F 00 41 F7

Return: This function returns 0 for success, or an error code for failure.

Notes: The function does not actually test the data for validity other than to send only valid Hex data. This means that it is the programmer's responsibility to ensure that SysEx messages are prefixed by 'F0' and suffixed by 'F7'. It also means that you can send any MIDI data with this function call, but it will be sent as a Windows long MIDI message.



Instrument Panel

The Instrument Panel allows you to interact with a particular synthesizer (or other MIDI device) by using Bank and Patch names instead of numbers. You can leave the Instrument Panel open while you work with other aspects of MIDI-OX: it is non-modal. To use it effectively you need to obtain an Instrument Definition file (.INS) for the device of interest. Definitions for a few instruments are supplied with MIDI-OX, including a generic one that can be used with any General MIDI device (GM.INS).

If you have a Cakewalk sequencer, you have access to a wide variety of Instrument definitions; you can find them in the Cakewalk Program Folder. You can also create new definitions using Cakewalk. If you don't have Cakewalk you can still get the full set of definitions by downloading a Trial version. Check this URL: <http://www.cakewalk.com>

Usage

When the Instrument Panel is opened the first time, it references a local Folder containing several Instrument definitions (.INS files). For this discussion, we will refer to the **Yamaha XG.INS** file, as it shows all of the interesting concepts. You can point the Instrument Panel at any other folder available to reference other INS files, or you can copy other INS files to your local MIDI-OX Instrument Folder. It will only reference INS files in a single folder though: any Favorite patch definitions must refer to an INS file in the current Instrument Definitions Folder, or they will no longer work (until you point to a folder containing the INS file they were based on).

Click on the INS File combo to drop down a list of Instrument definitions. Choose the **Yamaha XG.INS** file. In the **Instr** dropdown, choose the **Yamaha XG** instrument. You will notice that it selects a Bank Select method of **Normal**. This Bank Select method (BSM) indicates that the receiving device expects both a controller 0, followed by a controller 32 to effect a bank change, and it is a parameter that is part of the INS definition.

You can choose which channels Bank and Patch changes should be sent to (and listened to) in the right side of the panel. For now just select channel 1.

Click on the downward arrowhead to the right of the Bank button. A menu will open, listing available bank names within the XG definition. Choose the first bank (**XG Bank 0**). Click on the arrowhead to the right of the Patch button. A menu is presented containing the names of all 128 patches in Bank 0. Try selecting different patches. If you look back at the monitor window you will see that the Bank and patch changes have been sent out any open devices that are listening to the MIDI-OX Event object (see [MIDI Devices](#) > Routing). If you click on the wide Patch button it will re-send just the Patch. If you click on the wide Bank button, it will re-send both the Bank and Patch again.

Click on the Bank Menu button again, and this time choose XG Bank 16. Now open the Patch Menu button. You will see that the Patches are displayed with differing fonts: the slate blue font represents patches that are the same as the ones in Bank 0, while the ones in black represent patches that are different from Bank 0. This is because, in the instrument definition for XG, Bank 16 is **based on** Bank 0. Whenever a bank has a based on relationship you will see this effect.

Whenever MIDI-OX receives external Bank and Patch changes, the Instrument panel attempts to update it's view of the current bank and patch. This will also depend on which channels are selected (in the **Channels** box), and the buttons will display *Indeterminate* when differing Banks or patches are on different selected channels.

Favorite Patches

Whenever you find a patch you particularly like you can save it in a global, Favorite Patches menu. To do so, press the **Add To Favorites** button when the patch of interest is displayed. A menu item will be added to the Favorite Patches menu button, and added as the selected patch to the Favorites button. Information stored with each Favorite patch is the Instrument Definition File name, the Instrument, Bank and Patch. Selecting a Favorite patch sends it to the output ports and selects it as the current Instrument, Bank and Patch. To remove a favorite, bring it up (so it is selected) and then press **Remove Favorite**.

The Favorite Patches mechanism relies on the current **Instrument Definitions Folder** remaining static, so you may want to designate a single folder on your machine to contain all of your instrument definitions.

Scripting Interface

MIDI-OX now has a COM interface that can be accessed from any language that supports COM, including Visual Basic, C++ and Windows Script Host (WSH). WSH is part of Win98 and can be obtained from the Microsoft web site if you are using Win95. WSH comes with scripting engines for VBScript and JScript and third-party engines are available for other languages such as Python and Perl.

The MIDI-OX COM server (MoxKart.dll) provides access to various MIDI-OX and system properties (described below) as well as access to the MIDI stream. You can obtain MIDI data via a connection point sink or via polling in a loop. Your COM client can do what it likes with this data and then send it back to MIDI-OX for output to the selected MIDI ports. For example, you could change a particular controller to a different controller, or series of controllers.

We anticipate that the most common reason why someone would want to interact with MIDI-OX via the COM server is to obtain access to the MIDI stream. This would normally mean running in a loop, particularly for a script. In order to facilitate exiting this loop, we have added a menu item to MIDI-OX called **Exit WScript** under the Action menu item. Also, MIDI-OX will offer to send this menu command itself if you try to shut down MIDI-OX while the COM server is still attached.

Important: Your program should be designed to respond appropriately to this command. Failure to do so can cause unpleasant results. See below under **ShouldExitScript** and **OnTerminateMidiInput** to read about handling this situation.

The COM interface is described below. Where feasible, examples are given in VBScript. It should be relatively straightforward to translate this into JScript or Python, as well as providing enough information for the VB or C++ programmer.

An example script written in VBScript (VBSDemo.vbs) is provided with the install that demonstrates how to use the various properties, methods and events.

Getting Midi Input

The first thing to do when you want to communicate with MIDI-OX is to create an object. You can use the WScript object to both create an object and identify a prefix for a connection point sink. The sink will be called by MIDI-OX whenever MIDI data is available.

Example:

```
' Create object
```

```
Set mox = WScript.CreateObject("MoxKart.MoxWire.1", "OnTrigger_")
```

This statement creates an object and tells WScript to generate calls to a method named **OnTrigger_MidiInput()**. System Exclusive data will trigger an **OnTrigger_SysExInput()**. You create a subroutine in your script with that name to receive MIDI data. Data will not actually start flowing until you

- Make an attachment to a running instance of MIDI-OX
- set the **FireMidiInput** property.

To attach to a running instance you can first query to see how many instances are running, and then link to the one you want. You can also cause an instance to be launched (useful when no instances are running):

```
N = mox.NumberInstances
```

```
If n > 0 Then
```

```
If mox.AttachInstance( n ) = 1 Then ' i.e. success
```

```
Mox.FireMidiInput = 1 ' begins input
```

```
Mox.DivertMidiInput = 1 ' when set, routes all data
```

```
MsgBox "Press OK to end MIDI Loop" ' sits in a message loop
```

```
Mox.FireMidiInput = 0 ' stops MIDI input
```

```
Mox.DivertMidiInput = 0
```

```
End If
```

```
End If
```

```
' .....
```

' This is the function that gets called by MIDI-OX

```
Sub OnTrigger_MidiInput( timestamp, status, chan, dat1, dat2)
```

' ship it right back

```
Mox.OutputMidiMsg status + chan, dat1, dat2
```

```
End Sub
```

There are several things to notice about this code fragment:

- When the **DivertMidiInput** property is set, it routes all normal MIDI data through your **XXX_MidiInput()** sink. This is optional, however; and when the property is not set, you get a copy of the data.
- It is important to go into some kind of loop or put up a modal dialog or something to prevent your script from exiting, while you want to process MIDI data.
- Although the MIDI status and channel are separated in the generated **XXX_MidiInput** callback (event sink), **OutputMidiMsg()** expects a normal status byte to be supplied. The reason for this is that it is kind of a pain to split the status and channel (using VBScript anyway), but relatively trivial to sum them back together. Many times you might want them split out for processing.

Property and Method Reference

Application Hookup

GetAppVersion

Returns a string containing the version of MIDI-OX installed in the same directory as the COM interface object. No attachment is required to retrieve this value.

Example:

```
MsgBox "MIDI-OX Version: " & Mox.GetAppVersion
```

```
AttachInstance( nInstance )
```

Attaches the script object to a running MIDI-OX instance. The number passed in is *1 based* (the first instance should be attached as **AttachInstance(1)**).

Returns **1** when successful, **0** for failure.

Example:

```
N = 2
```

```
If Mox.AttachInstance( n ) = 1 Then
```

```
MsgBox "Attached 2nd Instance!"
```

```
End If
```

DetachInstance

DetachInstance will unhook an attached instance. The operation will be performed automatically when a script is exited or a new attachment is made (a single object is only allowed one attachment at a time).

Example:

```
Mox.DetachInstance
```

LaunchInstance

This method will launch a new instance of MIDI-OX, regardless of whether one is running or not.

Returns **1** when successful, **0** for failure.

Example:

```
If mox.LaunchInstance = 1 Then
```

```
MsgBox "Launched Instance"
```

```
End If
```

NumberInstances

This method returns the number of instances of MIDI-OX currently running.

Example:

```
N = mox.NumberInstances
```

Querying Devices

NumSysMidiIn

Returns the number of MIDI input devices installed in Windows.

GetFirstSysMidiInDev

Returns a VB string containing the name of the first MIDI input device defined to Windows.

GetNextSysMidiInDev

Returns a VB string containing the name of the next MIDI input device defined to Windows. Returns a blank string ("") when there are no more devices.

These methods can be used to return all the MIDI input devices installed. They do not require a MIDI-OX attachment (or even that MIDI-OX be running).

Example:

```
Str = "Sys MIDI In Devices: " & mox.NumSysMidiIn
```

```
StrWrk = mox.GetFirstSysMidiInDev
```

```
Do while strWrk <> ""
```

```
Str = str & vbCrLf & " " & strWrk
```

```
StrWrk = mox.GetNextSysMidiInDev
```

```
Loop
```

```
MsgBox Str
```

NumSysMidiOut

Returns the number of MIDI output devices installed in Windows.

GetFirstSysMidiOutDev

Returns a VB string containing the name of the first MIDI output device defined to Windows.

GetNextSysMidiOutDev

Returns a VB string containing the name of the next MIDI output device defined to Windows. Returns a blank string ("") when there are no more devices.

These methods can be used to return all the MIDI output devices installed. They do not require a MIDI-OX attachment (or even that MIDI-OX be running).

Example:

```
Str = "Sys MIDI Out Devices: " & mox.NumSysMidiOut
```

```
StrWrk = mox.GetFirstSysMidiOutDev
```

```
Do while strWrk <> ""
```

```
Str = str & vbCrLf & " " & strWrk
```

```
StrWrk = mox.GetNextSysMidiOutDev
```

```
Loop
```

```
MsgBox Str
```

NumOpenMidiIn

Returns the number of MIDI input devices opened in the attached MIDI-OX instance.

GetFirstOpenMidiInDev

Returns a VB string containing the name of the first MIDI input device opened in the attached MIDI-OX instance.

GetNextOpenMidiInDev

Returns a VB string containing the name of the next MIDI input device opened in the attached MIDI-OX instance. Returns a blank string ("") when there are no more devices.

These methods can be used to determine which devices are opened by the MIDI-OX instance. They can be compared against the system devices to determine the actual MIDI ID number (it is consecutive).

Example:

```
Str = "Open MIDI In Devices: " & mox.NumOpenMidiIn
```

```
StrWrk = mox.GetFirstOpenMidiInDev
```

```
Do while strWrk <> ""
```

```
Str = str & vbCrLf & " " & strWrk
```

```
StrWrk = mox.GetNextOpenMidiInDev
```

```
Loop
```

```
MsgBox Str
```

NumOpenMidiOut

Returns the number of MIDI output devices opened in the attached MIDI-OX instance.

GetFirstOpenMidiOutDev

Returns a VB string containing the name of the first MIDI output device opened in the attached MIDI-OX instance.

GetNextOpenMidiOutDev

Returns a VB string containing the name of the first MIDI output device opened in the attached MIDI-OX instance. Returns a blank string ("") when there are no more devices

Example:

```
Str = "Open MIDI Out Devices: " & mox.NumOpenMidiOut
```

```
StrWrk = mox.GetFirstOpenMidiOutDev
```

```
Do while strWrk <> ""
```

```
Str = str & vbCrLf & " " & strWrk
```

```
StrWrk = mox.GetNextOpenMidiOutDev
```

```
Loop
```

```
MsgBox Str
```

System Exclusive

SendSysExFile(FilePath)

This method will send a file containing SysEx out all ports attached by an instance, and which also are mapping the SysEx Port Map object (this is on by default when you open an output port in MIDI-OX). The file is expected to contain encoded binary SysEx data (not ASCII text). If you want to send a file containing SysEx represented as text, open the file and send it via the **SendSysExString** interface (below).

Example:

```
Mox.SendSysExFile "C:\Program Files\midiox\Syx\Sc55.syx"
```

SendSysExString(StrSysEx)

This method will send an ASCII string representing System exclusive data, out all ports attached by an instance, and which also are mapping the SysEx Port Map object (this is on by default when you open an output port in MIDI-OX).

Example:

```
Mox.SendSysExString "F0 41 10 42 11 48 00 00 00 1D 10 0B F7"
```

See Also: **GetSysExInput** (polling) and **XXX_SysExInput** (event sink) below

MIDI Input and Output

DivertMidiInput

DivertMidiInput is a property that works like a switch: when on MIDI Input is diverted through the script, when off a copy of the MIDI Input is supplied to the script. It can also be queried as if it was a method.

Example:

```
Mox.DivertMidiInput = 1 ' begin diverting input
```

```
If mox.DivertMidiInput = 1 Then ' query
```

```
MsgBox "Streams are diverted"
```

```
End If
```

FireMidiInput

This property can be set or queried and determines whether MIDI Input should fire a MIDI sink method defined in the script.

Example:

```
Mox.FireMidiInput = 1 ' begin firing MIDI events
```

```
If mox.FireMidiInput = 0 Then
```

```
MsgBox "Input will not cause trigger"
```

```
End If
```

ShouldExitScript

This property is usually queried in a loop, and may be set by MIDI-OX to inform the script that the user has chosen the **Exit WScript** menu item.

Example:

```
Do While mox.ShouldExitScript = 0
```

```
Msg = mox.GetMidiInputRaw()
```

```
ProcessInput( msg )
```

```
Loop
```

GetMidiInput

Retrieves MIDI data in the form of a comma delimited string from MIDI-OX. The layout format is as follows: timestamp,status,channel,data1,data2. Example: "65123,144,0,77,120". When no input is available an empty string ("") is returned. When System Exclusive data arrives, only the timestamp and a 0xF0 (240) status is received (other data is 0). To retrieve the SysEx string, you need to further call **GetSysExInput()** before calling **GetMidiInput()** again.

Example:

```
MsgStr = mox.GetMidiInput()
```

```
If msgStr <> "" Then
```

```
A = Split( msgStr, ",", -1, vbTextCompare )
```

```
Tstamp = Int(A(0))
```

```
Stat = Int(A(1))
```

```
Chan = Int(A(2))
```

```
Data1 = Int(A(3))
```

```
Data2 = Int(A(4))
```

```
If stat = &hF0 Then
```

```
    strSysEx = mox.GetSysExInput()
```

```
Mox.SendSysExString strSysEx
```

```
End If
```

```
End If
```

GetMidiInputRaw

This method retrieves input in the form of an encoded 32bit value. The timestamp is omitted, and the data is stored with the status in the least significant position (only the lowest 24bits are used to represent 3 MIDI bytes. The format is the same as that supplied to the Windows MME **midiOutShortMessage()** API. Example: if status=0x90, data1=0x60, data2=0x7F then the value would be encoded as: 0x007F6090. When System Exclusive data arrives, only the 0xF0 (240) status is received (other data is 0). To retrieve the SysEx string, you need to further call **GetSysExInput()** before calling **GetMidiInput()** again.

Example:

```
Msg = mox.GetMidiInputRaw()
```

```
Stat = msg And &h000000F0
```

```
Chan = msg And &h0000000F
```

```
Msg = msg \ 256 ' pull off stat
```

```
Dat1 = msg And &h0000007F
```

```
Msg = msg \ 256
```

```
Dat2 = msg And &h0000007F
```

```
If stat = &hF0 Then
```

```
StrSysEx = mox.GetSysExInput()
```

```
Mox.SendSysExString strSysEx
```

```
End If
```

GetSysExInput

This method retrieves System exclusive input after the other two polling methods (GetMidiInput and GetMidiInputRaw), have been advised that SysEx is available (in the form of an 0xF0 status value). The SysEx data is formatted into an ASCII string of hex digit bytes separated by spaces. The string is directly compatible with the SendSysExString function. If the SysEx message is longer than 256 data bytes, it is split, and you will receive one or more additional SysEx status bytes.

Example:

```
StrSysx = "F0 43 00 09 F7"
```

```
Msg = mox.GetMidiInputRaw()
```

```
Stat = msg And &h000000F0
```

```
If stat = &hF0 Then
```

```
StrSysEx = mox.GetSysExInput()
```

```
Mox.SendSysExString strSysEx
```

```
End If
```

OutputMidiMsg(nStatus, nData1, nData2)

This method sends a MIDI message out all MIDI ports that are mapping the channel. The **nStatus** parameter is a combination of the MIDI status and channel. Data1 and Data2 values depend on the MIDI message.

Example:

```
Mox.OutputMidiMsg 146, 78, 127
```

Connection Point Event Sinks

XXX_MidiInput(timestamp, status, channel, data1, data2)

This method is called by MIDI-OX whenever data arrives. In order to effect it, you must replace XXX_ with your prefix in both your subroutine definition and your **Wscript.CreateObject** call.

Example:

```
Set mox = WScript.CreateObject("MoxKart.MoxWire.1", "Test_")
```

...

Sub **Test_MidiInput(ts, stat, chan, dat1, dat2)**

Mox.OutputMidiMsg stat + chan, dat1, dat2

End Sub

XXX_SysExInput()

This method is called by MIDI-OX whenever System Exclusive (SysEx) data arrives. In order to effect it, you must replace XXX_ with your prefix in both your subroutine definition and your **Wscript.CreateObject** call.

Example:

Sub **Test_SysExInput(bstrSysEx)**

' Send it right back

Mox.SendSysExString bstrSysEx

End Sub

XXX_OnTerminateMidiInput()

This event will be triggered by MIDI-OX on the first MIDI message received after the user chooses the **Exit Wscript** menu item.

Example:

Sub **Test_OnTerminateMidiInput()**

MsgBox "MIDI Input Termination Received From MIDI-OX"

Mox.FireMidiInput = 0

Mox.DivertMidiInput = 0

End Sub

System Helpers

Sleep(msec)

Although a Sleep method is built into WSH 2.0, it was not available in earlier versions. Therefore we have added a wrapper around the Windows Sleep() function. The parameter specifies the number of milliseconds that the script should sleep.

GetSystemTime

Returns the current system time. It represents the number of milliseconds elapsed since Windows was started.

Example:

TNow = Mox.**GetSystemTime**

Mox.**Sleep(500)**

```
If tNow + 500 >= Max.GetSystemTime Then
```

```
MsgBox "A 1/2 second has passed"
```

```
End If
```
