MIDI Monitoring in a PC

Douglas A. Kerr

Issue 3
September 20, 2008

ABSTRACT

We may wish to be able to "observe" the MIDI message streams emitted by various electronic music programs in our computer, such as scoring programs or MIDI editors, perhaps in connection with trouble shooting. Various software tools ("MIDI monitors") allow these message streams to be captured, parsed, decoded, and displayed and/or printed for us. Their use is sometimes complicated by a gender issue at the computer's internal (logical) MIDI interface. In this article, we describe a typical MIDI monitoring tool (MIDI Ox); discuss the gender issue, and explain how it can be overcome; and give basic guidelines for the use of MIDI Ox.

INTRODUCTION

MIDI monitors

Software applications usually described as "MIDI monitors" take a MIDI message stream (which describes a musical performance), parse and decode it, and display it on screen (and/or log it to a file which can then be printed).

Some of these MIDI monitors also provide further functions, typically provisions for intervening in the message stream on its way to the destination "instrument" in order to modify the performance it describes, perhaps in order to try out different ways to exploit the capabilities of the instrument. For example, the score may currently call for a certain part to be rendered by an oboe, but we would like to hear how it would sound on an English horn without actually modifying the score to try that out.

A very capable MIDI monitor that I use is MIDI Ox, published by Jamie O'Connell. It is available here (free for noncommercial use):

http://www.midiox.com

This article discusses some of its features, how to put it into effect, and the basics of how to use it.

MIDI software applications

Typical MIDI software applications that might be the source of the MIDI message stream we wish to monitor include scoring programs, MIDI editors, and MIDI sequencers (players). These will be

characterized a little later when we discuss the overall architecture of a MIDI-aware computer.

The computer platform

All the discussions in this article are predicated on a PC-Windows platform.

MIDI PLUMBING

Introduction

Our deployment of a MIDI monitor to observe the MIDI stream generated by a MIDI application, such as a scoring program, is complicated by the way in which MIDI information is passed between entitles inside the computer, which I speak of as "MIDI plumbing".

The traditional MIDI interface

MIDI stands for Musical Instrument Digital Interface. The interface[1] in its traditional form (not found "inside" a computer) provides a vehicle for the transfer, between electronic music entities, of "MIDI message streams", which describe a musical performance, note-by-note.[2]

The original form of the interface is defined at the full range of layers: physical (connectors and pin assignments), electrical, transmission structure, transmission rate, message format, and message coding, and syntax.

A given entity ("MIDI device", in this context) may have an input port, an output port, or both, depending on its overall functionality. In the original physical/electrical form of the interface, input and output ports appear on separate but physically identical jacks on the MIDI devices. Any output port can be potentially connected, by way of a MIDI cable (which has identical plugs on both ends), to any input port. Of course for some physically-possible interconnections, depending on the functionality of the devices, no sensible communication might result.

The logical MIDI interface

Inside a PC computer, various MIDI-aware entities are linked by a "logical" interface. The concept is illustrated in figure 1. The message format and syntax follow those of the original interface.

---

[1] By custom, we speak of such an interface as "a MIDI interface", not as "a MIDI", even though the word "interface" is included within the meaning of the acronym.

[2] When the message stream for a performance is spoken of in its entirety, it may be called a "MIDI sequence".

On the right side of the figure, we show two illustrative "hardware" MIDI entities.[3] These are handled by the computer as I/O devices, and thus are provided with device drivers to allow access to them by MIDI-aware application programs.

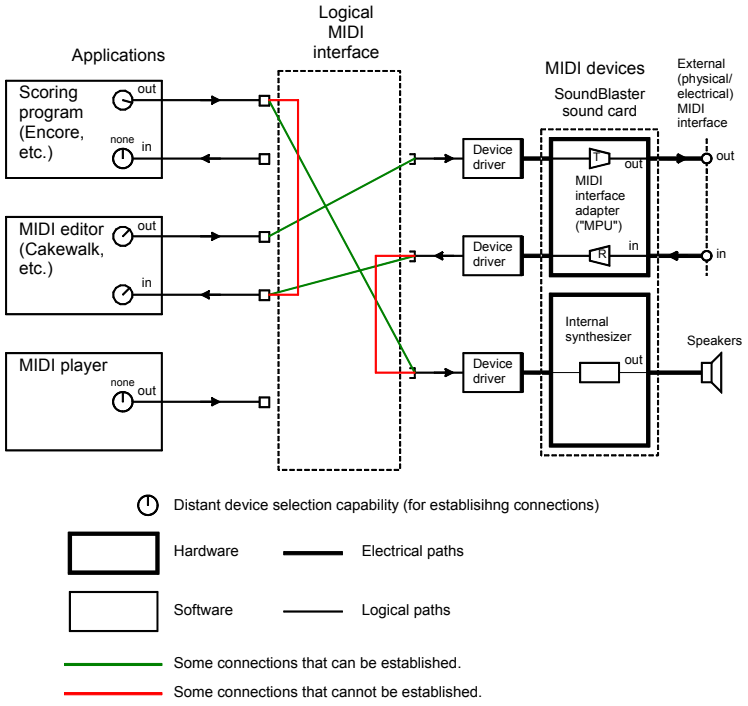

Figure 1. PC MIDI architecture

One device we show is a *synthesizer*, which takes a MIDI message stream describing a musical performance and (to the best of its ability) executes that performance, creating the sounds of (probably multiple) real or fanciful musical instruments by synthesis.

The second device we show is what can be called a *MIDI interface adapter*. This mediates between the logical MIDI interface inside the computer and an external traditional physical/electrical MIDI interface.[4] It might be used, for example, to connect the computer to an external synthesizer, or to an external keyboard (by which I don't mean a

---

[3] Of course today we will find microprocessors, running elaborate firmware suites, inside these units. But they overall are treated as hardware.

[4] We sometimes find this described as a "MIDI UART" in device lists, since it revolves around a Universal Asynchronous Receiver-Transmitter, which assembles (for output) and dissects (for input) the serial character format used for MIDI messages at the physical/electrical interface. It is also often referred to as an "MPU", since the first common unit of this type was the Roland MPU-401. Today, it is most often found as one portion of a sound card (a synthesizer being another portion).

keyboard instrument, capable of producing sound, but rather just a keyboard proper with a MIDI output interface).[5]

The figure also shows, on the left, three illustrative MIDI-aware software applications. One is a *scoring program*, such as Encore, Finale, or Sibelius. It does for musical scores what a word processor does for textual documents. It can "play" the score, meaning that a description of the musical performance implied by the score, in the form of a MIDI message stream, is sent out the MIDI output port of the application, perhaps to the internal synthesizer.

Most scoring programs can also accept a MIDI message stream, typically arriving from an external MIDI device by way of the MIDI interface adapter, to capture it in score form. (Perhaps the composer or arranger will use a MIDI keyboard to enter passages of the musical number into the scoring program, which sure beats entering the notes one at a time with a mouse!)

The second application is a *MIDI editor*, such as Cakewalk Home Studio, Sonar, or Master Tracks Pro. It allows the construction and modification of a musical performance—with emphasis on its representation as a MIDI sequence rather than in musical notation on a score, as in a scoring program. (There is nevertheless a lot of functional parallelism between these two kinds of applications.) As with the scoring program, it has both input and output MIDI ports, for essentially the same reasons.

The third application shown is what we would today call a *MIDI player* (although in fact its traditional name is *MIDI sequencer*). It reads from a computer file (a *standard MIDI file*, typically with filetype extension MID) a definition of a musical performance as a sequence of MIDI messages, and sends the messages in real time, through its output port, to (for example) an internal MIDI synthesizer. (Actually, both scoring programs and MIDI editors, when "playing" musical performances, are also acting the role of a MIDI sequencer.)

The input and output ports of these applications are all designed to link with MIDI devices in the computer through their device drivers, using a particular form of the computer's device driver protocol. The operation is directly analogous to a word processing program sending its output, via the device driver, to a printer, or receiving input, via the device driver, from the computer keyboard.

---

[5] Which might in fact be part of a keyboard instrument.

"Devices"

Note that in the discussion above we spoke of the entities on the right as "MIDI devices", since they are treated by the computer as "devices", and distinguished them from the entities on the left, which we described as "MIDI applications".

But outside the computer, in the world of the physical/electrical MIDI interface, we tend to call all the participating entities "MIDI devices", even those that are the functional analogs of the "MIDI applications" inside the computer.

This difference in the scope of the term "MIDI device" between these two contexts can cause confusion in discussions of MIDI message flow.

Establishing the connections

Inside our computer, how do we actually control the connection of outputs to inputs as needed for the desired flow of MIDI messages between MIDI devices and MIDI applications?

This is always arranged at the application end, whether it is the input or the output that is there.

Consider the scoring program Encore. If I wish to cause its MIDI output to go to the input of a certain one of my internal synthesizers, I open a "MIDI setup" dialog in Encore and open a dropdown menu for "Output". That menu will list all the existing MIDI devices (in the sense discussed above) having inputs. I choose the internal synthesizer. The result will be that the MIDI message stream emitted by Encore on its MIDI output will be led by the computer's operating system to the input of the device driver for the synthesizer (and thence to the synthesizer proper).

Similarly, if I want Encore's MIDI input port to receive the MIDI stream emitted by the receiving branch of my MIDI interface adapter (having arrived over a physical/electrical MIDI interface from, for example, a MIDI keyboard), I open a dropdown menu in Encore for "MIDI Input". That menu will list all the existing MIDI devices (in the sense discussed above) having outputs. I choose the MIDI interface adapter. The result will be that the MIDI message stream emitted by the receiving portion of the device driver for the MIDI interface adapter will be led by the computer's operating system to the MIDI input port of Encore.

So in all cases, the selection of the distant entity to which a connection is made is directed by the application involved, controlled by settings made on dialogs in that application. The little "knob"

graphics on the ports of the MIDI applications in figure 1 represent those "device selection" capabilities.

Impermissible connections

Of looming importance to us is the fact that:

- There is no way that an output port of a MIDI-aware application can be directed to send its MIDI output to the input port of another application, or vice-versa, nor for an input port of an application to be directed to "listen to" the output port of another application. These ports are only capable of linking with MIDI devices (via their device drivers). When connections are set up for application ports, ports on other applications do not appear on the selection list.

- There is no way that an output port of a device (that is, of its device driver) can be directed to send its output into the input port of another device, or a device input port directed to "listen to" an output port on another device. Connections with these ports can only be with application ports (set up at the application).

Thus, connections for the exchange of MIDI messages inside the computer can only be between an application port and a device port (and one of those ports must be an output and one an input).

The colored paths on figure 1 demonstrate this limitation.

The gender metaphor

The fact that we can only connect an **output port** to an **input port** is often spoken of as a matter of two distinct *genders* of the ports.

Similarly, the fact that, inside the computer, we can only interconnect **a port on a device** with **a port on an application** can be thought of as a second, and wholly separate, gender matter, which we might call *entity gender* to distinguish it from *port gender* matter discussed just above.

Thus, within the computer, the overall imperative is that **a port on an entity can only be interconnected to a port of the opposite *port gender* on an entity of the opposite *entity gender*.**

Note that the matter of *entity gender* is a creature of the asymmetrical nature of the device driver-application interface in the computer. It does not flow from any generalized functional or semantic dichotomy.

Outside the computer, in the world of the physical/electrical MIDI interface, the concept of *port gender* applies, but there is no inherent concept of *entity gender*. We can interconnect any output port with

any input port, and data can flow.[6] Of course, whether or not this will be a meaningful interchange is another matter altogether.[7]

CONNECTION OF A MIDI MONITOR

Introduction

The type of MIDI monitor we consider here is a software application. Its traditional use is to accept a MIDI message stream arriving from outside the computer, entering through a MIDI interface adapter, and display it. In this situation, its input port is linked in the now familiar way with the output port of the adapter, as shown in figure 2. Of course, the two linked entities here are of opposite entity gender.

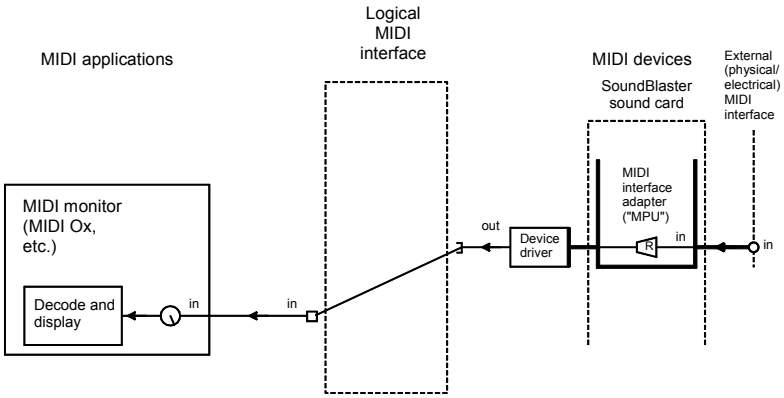

Figure 2. MIDI monitor with "external" message source

But in the case of interest in this article the source of the MIDI stream to be examined is another MIDI application in the computer, such as a scoring program. Thus we need the MIDI message stream emitted by the output port of the scoring program (while "playing" a score) to be carried to the input port of our MIDI monitor.

The entity gender limitation

As we have recently seen, the output port on the scoring program is only able to choose from among MIDI devices—not other applications—as a destination for its output, and the input port of the

---

[6] We can in fact physically connect an output port with another output port, or an input port with another input port, owing to the nature of MIDI cables. But this will be futile at all levels: no messages can flow through such a connection.

[7] We might, for example, connect an output port delivering a MIDI message stream describing a musical performance to an input port expecting to receive only MIDI timing messages. Messages will flow, but the interchange will be semantically futile. But this is not a result of entity gender conflict, merely of functional incompatibility.

monitor program is only able to choose among MIDI devices—not other applications—as a source for its input. Thus neither of the ports of interest is able to choose the other as its correspondent—the interconnection we need cannot be established using the inherent capabilities of the entities and the computer.

This limitation is a manifestation of the structure of *entity gender* created by the computer's reliance on the asymmetrical "device driver" architecture for the interchange of MIDI messages.

Circumventing the gender limitation

In order to circumvent this entity gender limitation, we must introduce a special software "intermediary", a *device side MIDI loopback driver*. From a software standpoint, this is organized like a bidirectional device driver, but in fact is not associated with any hardware device. Rather, it just (in effect) takes any MIDI message directed into its input port (from the output port of some MIDI application) and sends it out verbatim over its output port (to the input port of some MIDI application).

Then, we can establish the needed routing between the MIDI output of our scoring program and the MIDI input of our MIDI monitor as shown in figure 3.

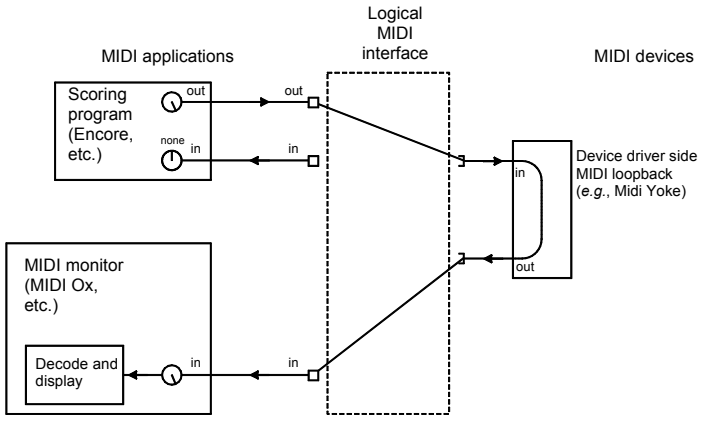

Figure 3. MIDI loopback to monitor

Establishing this routing involves the following:

- On the scoring program, we set its MIDI output selection to the loopback driver, which will have some easily-recognizable name—in this case, that name represents the input port of the loopback driver.

- On the MIDI monitor program, we set its MIDI input selection to the loopback driver—in this case, the name represents the output port of the loopback driver.

MIDI Yoke

A handy device-side MIDI loopback driver is **MIDI Yoke**, published by Jamie O'Connell, who also publishes the **Midi Ox** MDI monitor application. It is available here (free):
http://www.midiox.com

An installation of MIDI Yoke (it needs to be installed just like any other device driver) actually installs several instances of the driver, each of which are independent loopback facilities (with distinct names). For some complicated setups (beyond what we describe here), more than one loopback driver might be simultaneously needed.

You can choose how many instances are to be installed, up to 16. Having too many isn't necessarily harmless, since some MIDI applications have a limit as to how many MIDI devices they can enumerate on their "port connection" menus. If you use up the quota with too many instances of a loopback driver, some other devices might be invisible and thus inaccessible.

The installation of MIDI Yoke is covered in Appendix A.

But now I can't hear it

Now that we have redirected the output of Encore away from our synthesizer and to MIDI Yoke instead, allowing it to ultimately reach the MIDI Ox Monitor, we no longer hear the performance. But we may wish to still hear it so we can "follow the music" as we watch the monitor.

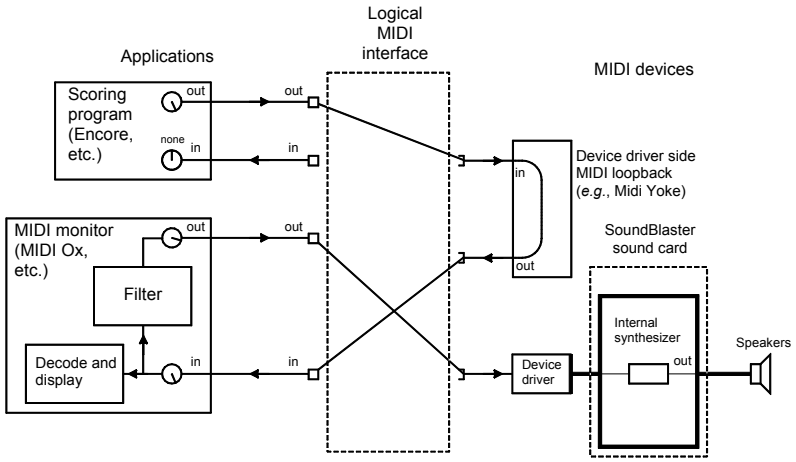

Figure 4. Complete monitoring setup

Fortunately, MIDI Ox provides an additional feature that we can employ to overcome this problem. It allows the incoming MIDI stream to be sent out its MIDI output. A main reason for this is that this "through" MIDI stream can be filtered and modified in many ways, on the way to, for example, a synthesizer, as might be useful for special applications.

But we can just use that path without any such intervention to lead the MIDI stream, unaltered, to our synthesizer for "audition" purposes.

Figure 4 shows the complete arrangement.

The configuration of MIDI Ox and the reconfiguration of our "source" application (such as Encore) to provide this arrangement is covered in Appendix B.

MIDI MESSAGES

Before we continue, I will give a brief review of the basic structure of a MIDI message so that the material on MIDI message stream monitoring can be put into context.

MIDI messages describe, in real time, a musical performance, in note-by-note form.

Each MIDI message consists of one, two or three bytes. Only seven bits of the bytes actually carry data. The highest order bit is a flag that allows the receiver to synchronize with and parse the message stream: it is always "1" for the first byte of a message and "0" for all later bytes.

The first byte is called the *status byte*. In *channel messages*, it identifies in its three high-order data bits the message type. In effect, this gives the "verb" of the message: what the message tells the recipient to do. For each channel message type, there will always be either one or two data bytes to follow.

In *channel messages*, the lower-order four bits of the status byte carry the *MIDI channel number*, sort of a routing tag for the message. This tag system essentially creates 16 "virtual channels" (spoken of as channels 1 through 16, although the encoded values are 0-15) "through which" messages can distinctly travel.

This "channelization" allows messages to coexist in a single stream that have up to 16 different "destinations", either in the sense of one of several physical devices being fed the same message stream, or (more commonly) one of several "compartments" of a "multi-tasking" synthesizer, each acting as one "instrument" in a musical ensemble.

In *system messages* (for which there is no channel concept), the three high order data bits are always 111, and the four low order data bits give the message type. For each system message type, there will be no, one, or two data bytes.

In messages of either class, the data byte(s) normally each carry a numerical parameter of the message, which define the details of the function to be performed (such as "what note is to be sounded" and "how loud", or "what should be the overall volume level for the instrument controlled over this channel").

For some message types, there are two data bytes that collectively carry a single 14-bit parameter. (For more on this, see Appendix C.)

As a metaphor, the status byte of a fanciful channel message might say "jump" and "who should jump" (that is, "people listening to channel 5 for their instructions"). The two status bytes might say "how far" and "in what direction".

USING MIDI-OX

Introduction

Now we are ready to actually use MIDI Ox to observe the MIDI message stream coming from out source application.

Midi Ox is very rich in features, going far beyond its basic monitoring function. However, in this article, we will mostly limit ourselves to that basic function.

Basic display modes

Keeping in mind the "MIDI through" capability we alluded to earlier, it is not surprising that Midi Ox will allow us to separately monitor (parse, decode, display) the MIDI stream arriving at its input port and the MIDI stream being sent out.

The default display is of the output stream. In fact, that display window is always active (although it can be minimized).

If we want to monitor the input MIDI stream (which is ordinarily actually our interest in this article), we need to open the **input monitor** window. To do that, we select **View|Input monitor**. A second display window will then open.

We can then "close" the output monitor window. The actual result is only that it is minimized, but the minimized window will usually be hidden behind the other monitor window (Input).

Of course, since we presumably have Midi Ox set to pass all received MIDI messages to its output verbatim (even though they might not be directed from there to anyplace), the output monitor display window will actually show us what we are interested in (most of the time). Thus we might choose merely to rely on it, and not open the input monitor window at all. (I recommend not to do so, since some confusion can arise under certain situations.)

If by any chance we are unable to see the MIDI stream of interest, it might be because:

- The "through" feature has been disabled, and
- The only visible display window is the **Output monitor** window.

The easiest way to re-establish the basic through functionality (without having to learn the rather "rich" port management interface) is to repeat the configuration procedure in Appendix B.

The monitor display

The monitor display comprises one line for each MIDI message in the stream, listed in order of their arrival. For each message, we see the following:

- TIMESTAMP. This is the arrival time of the message, in milliseconds, on an arbitrary scale (it starts when the application is started). It appears that the number can run to nine digits (almost a million seconds, over 11 days), but it restarts if the application is closed and restarted.
- IN. The device from which the message arrived (through the MIDI Ox input port)—the proximal source of the message. (The numbers of the different possible devices are shown on the **Options|MIDI Devices** dialog). In the case of the output display, we may see internally-initiated messages that did not arrive through the input, and a mnemonic code is used in the IN field to indicate such an origin.
- PORT. (Output monitoring display only.) The device to which the message is sent via the MIDI Ox output port—the proximal destination of the message. Again, the number assignment is shown on the **MIDI Devices** dialog. (On the input monitor display, this field exists for format consistency, but carries just "--".)
- STATUS. The numeric value of the first byte of the message, the *status byte*. Recall that, except for system messages, it indicates both the type of the message (its "verb") and the MIDI channel number, and its value reflects the immutable "1" value of the most

significant bit of a status byte. Both the channel number and the message type are shown separately in other fields of the display line. In system messages, the status byte only carries the message type, and the channel filed shows "--"..

- DATA1 and DATA2. The numeric values of the data byte(s) of the message (some message types have only one). These carry the parameters of the message, and their significance of course varies with the message type. For messages with only a single data byte, the DATA2 field shows "--".

  For some message types, the two data bytes collectively give a single 14-bit parameter value. This value is not directly displayed in the listing. Midi Ox includes a calculator that will determine the value from the DATA1 and DATA2 byte values (it is called the "NPRN calculator".[8] In it, the DATA1 and DATA2 values are entered as "LSB" and "MSB", respectively (note the "reversed" order). The calculation can also be easily done manually. The procedure is given in Appendix C.

- CHAN. This is the channel dictated by the channel tag in the status byte (for channel messages only). The channels are identified as 1 through 16.

- NOTE. For **note on** and **note off** messages, the affected note (whose "note number" is given by the DATA1 byte) is shown in enharmonic notation with an octave indicator, such as C# 3.

  Note that different schemes of octave notation are used in various contexts. For example, "middle C" is often designated as C 3, or as C 5, or as something else. The reference point for the octave indication in the display in Midi Ox can be changed in the Options|General dialog.)

  A note number of 60 (3Ch) indicates middle C. This equivalence is not affected by the convention used to "display" the octave of the note—middle C is the same note (same pitch), and has the same note number, whether we choose to display it as "C 3" or "C 5".

  For other message types, this field just shows "--".

- EVENT. This is the type of message, an indication of the action that it prescribes. Examples are **note on**, **note off**, **PC** (program change), **CC** (controller change), and so forth.

---

[8] NPRN stands for *non-registered parameter number*. It is one of the quantities whose value is represented in 14 bits (although not from the two data bytes in a single message—from a data byte in two "paired" messages).

For program change messages, the name of the instrument ("patch") implied by the new program number is displayed, normally based on the General MIDI assignment. The new program number itself can be read in its coded form as the DATA2 value. Note that the coded values run from 0 through 127. But the program numbers are often (but not always) written or displayed in applications on the basis of 1 through 128.

The display of the patch name can be disabled.

For **CC** messages (actually a family of message types), the name of the specific "controller" involved (such as **volume**) is displayed. It is designated by the value of DATA1. The parameter for the prescribed action is carried by the DATA2 value.

The monitor entries are color-coded in terms of message type or type family. We can choose to have them color coded instead by channel number.

Capturing the message stream

The message stream appearing at the input to Midi Ox (and/or the one being sent out the output) will only be captured, parsed, decoded, and displayed if the "record" function is on, which is indicated by the REC notation at the bottom of the main window being highlighted. It is normally on by default. If the record function has somehow been turned off, click on REC to turn it on.

Clearing the display

Either monitoring display can be cleared by making its window active and selecting **Actions**|**Clear Monitor** (or pressing the "swash X" icon on the toolbar).

Logging the message stream

The journal of messages we see in the output monitor window can also be written to a text file for later examination or for sending to a printer. (We cannot do this for the input monitor journal.)

To enable the logging function, select **File**|**Log**. A dialog will open. Check **Enable Logging**. The path and filename of the log text file can be set on this dialog.

When logging is in effect, LOG will be highlighted at the bottom of the main window. The function, however, cannot be enabled or disabled by clicking on that indication.

If we wish, the log file can be read (in our system default text editor) by clicking the **View Log** button in the log dialog, but the file must first be closed (by unchecking **Enable Logging**).

Since it is the output monitor display that is logged, if we want to log our MIDI input stream, we must be certain that the MIDI Ox "through" feature is enabled (as discussed earlier) so that all incoming MIDI messages are sent to the output port (verbatim, if we haven't established any "filtering" or "remapping").

The default format for the log file directly corresponds to the monitor display screen format. Two other forms are available:

- "MIDI to text", in which the messages are given in mnemonic form, sort of a MIDI "assembly language". This language can be used to "hand code" a MIDI sequence. The code can be "compiled" to produce a standard MIDI file.

- A verbatim representation of the MIDI byte stream, with the bytes shown in hexadecimal format, the messages (two or three bytes) separated by spaces.

The Keyboard

MIDI Ox allows us to use the computer's QWERTY[9] keyboard to send MIDI note messages out the MIDI Ox output port (in our setup, to our internal synthesizer). Thus we can "exercise" our synthesizer for investigatory work, or play a little concert for visitors without hauling out our MIDI keyboard.

The bottom row of keys (Z-/) represent the white keys of a piano keyboard from C2 through E3 (in the notation where middle C is C3). The keys in the row above (A etc.) are the corresponding black keys (and of course some are inactive, where there are gaps between the black keys in a regular piano keyboard layout).

The top row of alpha keys (Q-]) represent the white keys from C3 (middle C) through G4). The keys in the numeric row above are the corresponding black keys.

The function is polyphonic; that is, two or more keys can sound simultaneously.

By default, the note messages emitted by this keyboard are tagged for MIDI channel 1, but this can be changed. They have a default velocity ("loudness") parameter of 100, but this can be changed. We can also

---

[9] *Ou* AZERTY, *n'est-ce pas*, *oder* QWERTZ, *nicht war*?

change the octave range, or set the keyboard so that it can produce various preset chords.

We can also, by various fancy key manipulations, manually send out many types of MIDI control messages. The details are beyond the scope of this article, but are well documented in the extensive Midi Ox **help** facility (enter “keyboard” in the help index field and select the top entry).

To activate this function, select **Actions**|**Keyboard**, or click on **KYB** at the bottom of the main window. The main MIDI Ox application window must have the focus for this function to work.

MULTI-CLIENT PORTS ON MIDI YOKE

This matter isn’t pertinent to the specific setup so far described in this article, but can be of importance when setting up more elaborate connection schemes among MIDI entities in a PC.

Normally, the ports of the device drivers we have been discussing are “single client” ports. That means, for example, that if we have set an input port of a certain MIDI application to “listen to” the output port of some MIDI device driver, we then cannot have another application input port also “listen to” that same device driver port. If we try, we will get an error message (either when we try and establish that second connection, or when the second application actually attempts to “open” the device driver port).

Similarly, if we have set an output port of a certain MIDI application to direct its output to, say, the input port of a device driver, we then cannot then tell an output port on another application to also direct its data to that same port.

This is not an absolute limitation of the device driver interface scheme. That scheme admits of “multi-client” port designs. Rather, it is a limitation the designers of specific device drivers have adopted to simplify the driver code. (And in the case of the second example, allowing two applications to contemporaneously send to the same device could lead to various semantic complications, a problem that is mooted if the port will just not allow it.)

However, the input and output ports on an instance of MIDI Yoke are all of the “three-client” form. That is, we can have up to three MIDI application input ports set to take their data from the output port of a single MIDI Yoke instance.

And we could have up to three MIDI application output ports set to direct their data to the input port of a single MIDI Yoke instance. (In that case, if more than one application is actually sending MIDI

messages at the same time, MIDI Yoke interleaves the messages into a single output stream. Whether that would make any sense to the ultimate recipient of the stream is another matter altogether.)

But that arrangement can be handy when we want to send to a device from one application for a while, and then from another application, without having to rearrange the connections.

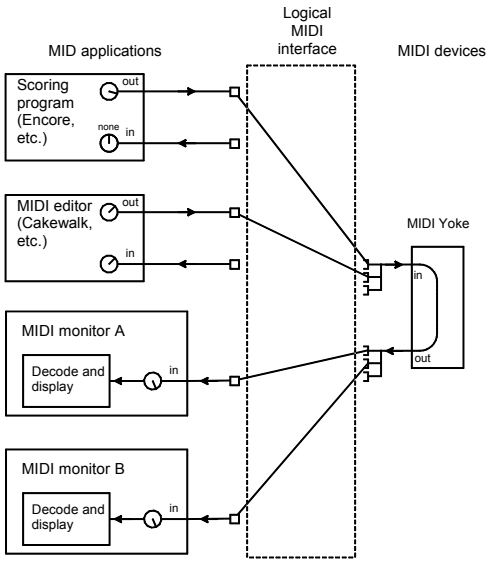

Figure 5. Exploitation of multi-client ports

For example, in our situations, we might have both our scoring program and our MIDI editor send into the same MIDI Yoke instance, which in turn sent to our MIDI monitor. Then we could send a MIDI sequence from either application to the monitor, for inspection, without having to do any "switching". This might be handy, for example, to compare MIDI sequences from the two applications.

And of course, we could have two different MIDI monitor programs take their data from the output port of one MIDI Yoke instance, if we wanted to use both to inspect the same MIDI message stream (perhaps they provided complementary analytical capabilities).

Note that the separate "branches" of a given device driver port are not separately listed in the port selection dialogs of applications. The operating system protocol takes care of the matter automatically (if the device drivers have the capability).

Figure 5 shows both these aspects in use.

#

APPENDIX A

Installation of MIDI Yoke

MIDI Yoke is a quasi device driver for PC computers, providing a "device-side loopback capability". This allows us to route MIDI messages from one MIDI application to another (ordinarily, a MIDI application can only route to or from a MIDI device in the computer).

Midi Yoke is published by Jamie O'Connell, also the publisher of the MIDI monitor Midi Ox. The program can be obtained here:

http://www.midiox.com

(Choose Midi Yoke from the navigation panel.)

There are different versions for different vintages of the Windows operating system., We will discuss here the version suitable for Windows NT and XP. We have no firm information as to whether it will work under Windows Vista.

The easiest installation is with the form of the distribution file that works with Microsoft Installer (MidiYokeSetup.msi). We will describe this installation process.

After downloading that file to an appropriate directory, in a file manager or Windows Explorer, right click on the file and choose Install. Follow the onscreen directions. At one point, you will be presented with a configuration window. It allows you to choose the number of instances of the loopback driver that will be installed (the box is labeled "Number of Ports"), up to 16 (8 is the offered default). Unless you know that you will need a lot of instances, I suggest you choose to install two instances.

This window also allows you to vary the strategy the driver will use to thwart "MIDI feedback", the recirculation of MIDI messages resulting from the improvident deployment of MIDI Yoke. Unless you are familiar with this very esoteric topic, I suggest you leave the default strategy in place.

Once the installation is completed, then all the MIDI Yoke instances you have installed will appear in the device lists of the input and output port MIDI device selection dialogs in all MIDI applications. The several instances have distinct names, such as (for the input aspects, as appear on output port device selection dialogs) **Out to MIDI Yoke: 1** and **Out to MIDI Yoke: 2**. (The name preambles are intended to remind the user, looking at the device list, what kind of connection is being established. Most device names don't have a preamble like that.) On

application input port device selection lists, the names of the MIDI Yoke output aspects are (for example) **In from MIDI Yoke: 1** and **In from MIDI Yoke: 2**.

If you later wish to change the number of installed instances (or the feedback avoidance strategy), you will need to access the driver configuration window, thus:

1. In the **Windows Control Panel**, select **System**
2. Select the **Hardware** tab.
3. Select the **Device Manager** button.
4. Expand **Sound, video and game controllers**
5. Right click on **Legacy Audio Drivers**
6. Select **Properties**.
7. Select the **Properties** tab.
8. Expand **MIDI Devices and instruments**.
9. Select **MIDI for MIDI Yoke NT driver**.
10. Select the **Properties** button.
11. Be sure the **General** tab is selected.
12. Select the **Settings** button.
13. You now will have the MIDI Yoke configuration window.
14. You can change the number of instances of MIDI yoke to be in effect (the box is labeled **Number of Ports**) or the MIDI Feedback strategy.
15. Exit all the way. The changes will go into effect at the next Windows restart.

You can see why it is worthwhile to choose the proper number of instances during initial installation!

#

APPENDIX B

Configuring Midi Ox

Midi Ox is a MIDI monitor that allows us to intercept the stream of MIDI messages emitted by a MIDI application, parse and decode it, and display it on screen (and/or write it to a text file for later examination of printing). This appendix describes how to configure Midi Ox for the basic form of this activity. It also describes how to arrange for the MIDI messages to be properly routed for the activity.

Midi Yoke is published by Jamie O'Connell. The program can be obtained here:

http://www.midiox.com

Installation is straightforward.

We assume that there is at least one instance of the device-side MIDI loopback driver, Midi Yoke, installed.

Configuring the source MIDI application

First, we must arrange for the source MIDI application (perhaps a scoring program, MIDI editor, or MIDI sequencer) to send its MIDI messages to Midi Yoke rather than (for example) directly to the internal synthesizer.

1. Open the **MIDI Setup** (or equivalent) dialog in the application. Look at the panel for **MIDI output**. The box will probably show a destination such as **SoundBlaster Deluxe MIDI Synthesizer**. Note what that is (we'll need it later).

2. Take the dropdown for the box. You should see a list of MIDI devices with inputs, which should include one or more instances of MIDI Yoke. Choose one for this project (we will assume it is **Out to Midi Yoke: 1**) and select it in the box.

The result is that the MIDI messages from the application will be directed into the MIDI Yoke loopback driver, from which they will be available to our MIDI monitor (Midi Ox).

Configuring Midi Ox

1. In MIDI Ox, select **Options|MIDI devices**. In the **MIDI Inputs** pane, select **In from MIDI Yoke: 1**. Be sure that there are not multiple selections in effect (normal Windows selection techniques work here). It is not necessary to "execute" this selection—whatever

selection is in place will come into effect as soon as the dialog is OKed.

The result will be that the MIDI messages sent by the source application to MIDI Yoke will now be taken into Midi Ox for parsing, decoding, and display.

2. In the **MIDI Outputs** pane, select **SoundBlaster Deluxe MIDI Synthesizer**, or whatever device had been the original destination of the source application.

3. Check **Automatically attach Inputs to Outputs during selection**.

   The result of these two last two steps will be that the MIDI messages originally sent by the source application, and received into MIDI Ox, will also be sent out of MIDI Ox to the synthesizer (verbatim, if we don't put into effect any "filtering"), so we can still hear the performance.

4. **OK** the dialog to put the settings into effect.

5. You will probably want the display of the values in the MIDI messages to be in decimal (rather than hexadecimal) form. To arrange for this, select **Options|Data display** and be sure that **Monitor Input: Hex** and **Monitor Output: Hex** are both unchecked. If you have to uncheck them, you will have to visit the menu separately for each of them.

#

APPENDIX C

Decoding 14-bit MIDI message paramaters

Certain MIDI message types (such as the Pitch Bend message) have a 14-bit parameter value. Seven bits of this value are carried by each of the two data bytes. Bits 0-6 of the parameter value are carried by bits 0-6 of the first data byte, while bits 7-13 of the parameter value are carried by bits 0-6 of the second data byte.

Note that in any case, only 7 of the bits of each data byte carry data; bit 7, the “most significant” bit, is always “0”, a flag indicating that this byte is not the first byte of a message. Thus, the values of the data bytes will never be greater than 127 (decimal) or 7F (hexadecimal).

A 14-bit parameter value can never be greater than 16383 (decimal) or 3FFF (hexadecimal).

MIDI Ox does not display for us, for such messages, the parameter value—just the values of the two data bytes. But, from those values, we can easily determine the parameter value.

Let X represent the value of the first data byte and Y the value of the second. Then V, the parameter value, is given by:

$$V = 128Y + X$$

Note that this is valid for work either in decimal or hexadecimal notation. The factor 128 (decimal) is numerically the same in either case, but of course if we are working in hexadecimal, perhaps using a hexadecimal calculator, we must enter that constant as 80h.

#