MIDI Plumbing Inside a PC

Douglas A. Kerr

Issue 2
September 20, 2008

ABSTRACT AND INTRODUCTION

Various "electronic" music activities inside a PC involve the transmission between entities of "MIDI message streams", streams of coded messages that describe a musical performance. The flow of these streams from one entity to another is not over the traditional MIDI electrical interface, but rather over a "logical interface", similar to the ones between applications and I/O devices such as printers and keyboards (the overall arrangement of which can be described as "MIDI plumbing"). Because of the asymmetry of this interface, there arises the concept of *entity gender*, which may impede us from arranging certain useful information flows. In this article, we describe this architecture, explain the limitations it imposes, and discuss various ways to circumvent those impediments.

MIDI—THE MUSICAL INSTRUMENT DIGITAL INTERFACE

The basic concept

The acronym MIDI—Musical Instrument Digital Interface—is used to identify a range of related concepts in the field of electronic music.

It initially described just what it seems to, a (physical) interface through which we could transmit to a musical instrument (conventional or "synthesized"), in real time, the note-by-note details of a musical performance, so as to automate that performance. The interface is defined at a number of levels: physical, electrical, transmission format and rate, and syntax at several levels. Such an interface is usually referred to as a "MIDI interface" even though the word "interface" is in fact covered by the acronym itself.

A complementary use is to convey from a MIDI-aware keyboard (and I don't mean here a "keyboard instrument", which actually produces sound—just the keyboard itself[1]) a "performance" done on that keyboard to a device that would store it so that it could later be sent to an instrument to recreate the performance.

A later, closely related development is the "standard MIDI file", a standardized computer file format for storing a complete description of a musical performance in a way that is directly relatable to its

[1] That keyboard might, of course, be a part of a "keyboard instrument".

real-time description as a MIDI message stream. For the most part, the file consists of a transcription of a sequence of MIDI messages with time tags.

A device functionally called a “MIDI sequencer” (we would today more commonly say “MIDI player”) can read such a file and from it generate a MIDI message stream that is sent (over a MIDI interface) to a MIDI instrument.

MIDI devices

In the original world of the MIDI interface, the term “MIDI device” is often used to describe any entity that could send and/or receive MIDI message streams. We’ll see later that a slightly different usage comes into play inside a PC.

Directionality and connectors

Some MIDI devices emit MIDI streams, some receive them and act on them, and some do both. A full-blown MIDI interface thus has two paths, one for information passing from device A to device B and one for information in the opposite direction. From a physical standpoint, these appear on separate physical connectors at both ends. Normally, the MIDI input and MIDI output ports appear as “chassis-mounted” jacks (female pins) on the MIDI device.

MIDI cables, having identical male-pin plugs on both ends, are used to connect the output of one device to the input of another[2]. One can of course use a cable to connect the MIDI output of one device to the MIDI output of another device, but nothing will happen.

Since we connect one **output** to another **input**, the matter of “tagging” cables to keep track of them in a complex installation can be rather tricky.

We can think of the complementary nature of input and output ports as a matter of *port gender*.

Peerage

The definition of the interface itself involves no implied concept of “master/slave”. All output ports are electrically the same, and all input ports are electrically the same (but different from input ports—input and output ports are electrically complementary.)

---

[2] The wiring is “pin-to-pin”.

Of course, in an overall operation of a "setup" of MIDI devices, there will often be a functional asymmetry, sometimes properly describable as a master-slave relationship—but it is not a creature of the interface definition.

INSIDE THE PC

Introduction

Our objective here is to talk about the flow of MIDI streams inside a PC.[3] In this environment, we can encounter a number of MIDI-aware entities (I avoid the term "device" for a reason that will shortly be abundantly clear.) These are often functional analogues of familiar "external" MIDI devices.

Important entities include (some names are in bold for a reason that will appear shortly):

- **Synthesizer**. This accepts a MIDI stream and generates the sound of the performance it describes, often by emulating the sound of traditional musical instruments. The resulting sound is typically amplified and sent to loudspeakers. The synthesizer may be part of an accessory "sound board" or embedded on the computer's "motherboard". We consider the synthesizer to be "hardware", although today it inevitably involves one or more microprocessors running elaborate suites of firmware.

- **MIDI interface module**.[4] This hardware device mediates between a classical MIDI interface (physical/electrical) and the interior of the PC. It would typically be used to link "external" MIDI devices (a MIDI-aware piano, for example, or a MIDI-aware keyboard) to software MIDI entities inside the computer. The interface module may be on a distinct accessory board, part of an accessory "sound board", or even embedded on the computer's "motherboard".

- MIDI sequencer. This gets its traditional name in that it emits a sequence of MIDI messages to represent a musical performance. Today we may more likely call it a "MIDI player". It typically reads a MIDI file and from it generates a MIDI message stream describing the performance, to be passed to a MIDI instrument for rendition. The kind we find inside a PC is a software application.

---

[3] Note that the concepts discussed here are generally applicable to implementation on Mac computers, but some of the details, implications, and terminology will differ.

[4] Such a device is sometimes referred to as an "MPU", a reference to one of the first popular items of the type, the Roland MPU-401.

- Scoring program. These software programs are to a musical score what a word processor is to textual documents or a CAD program to engineering drawings. Common scoring programs include Encore, Finale, and Sibelius. These programs allow us to construct a musical score following the complex syntactical and graphic conventions that are involved. Scoring programs typically allow us to "play" (or "audition") a score by emitting a stream of MIDI messages to a MIDI instrument. They may also be able to accept a MIDI stream from, for example, a keyboard, and record the performance in score form.

- MIDI editor program. These software programs allow sophisticated manipulation of MIDI files representing performances, in fact even allowing the performance to be wholly created. Common MIDI editors are Cakewalk, Sonar[5], and Master Tracks Pro. Like scoring programs, these can then emit a MIDI stream describing the "composition" for audition purposes. They can also accept a MIDI stream from, for example, a keyboard, so as to capture the performance it represents as part of the "composition" being constructed.

We will encounter other, more specialized, MIDI entities a bit later.

The "logical" MIDI interface

Inside the PC, MIDI streams are passed between the various entities over what we can call the "logical MIDI" interface. This is wholly software oriented. It revolves around the "device driver" architecture and syntax of the computer, administered by the operating system. It wholly parallels the architecture used for applications to interface with familiar I/O devices such as printers, the computer keyboard, and so forth.

The entities whose names appear in bold in the list (*e.g.*, synthesizer) of entity examples are, in general, basically hardware devices (they may, however, have a large software or firmware content), and as such, must be supported by *device drivers* to be accessible to software applications. Those drivers have to be installed, and they are "loaded" every time the computer starts.

The other entities in the list (*e.g.*, MIDI sequencer/player) are software applications. They are only "loaded" when they are "launched". Just

---

[5] In fact, programs like Sonar go far beyond the MIDI editor capability, allowing the manipulation and integration of both "note-oriented" (MIDI) music descriptions and actual sound files, with provisions for editing both.

as a word processor application is able to send information to a printer only by way of the printer driver (using an appropriate protocol under the auspices of the operating system), these MIDI applications are able to access a device such as an internal synthesizer by way of its device driver.

Nor surprisingly, in the context of MIDI life inside the PC, such things as a synthesizer or MIDI interface module, which are "devices" from the computer architecture perspective, are referred to as "MIDI devices". We don't use that term for the MIDI applications (which are not "devices" from the perspective of computer architecture).

However, the external, physical analogues of these MIDI applications **would** be called "MIDI devices". This difference in notation is the cause of confusion when discussing the concepts in this article.

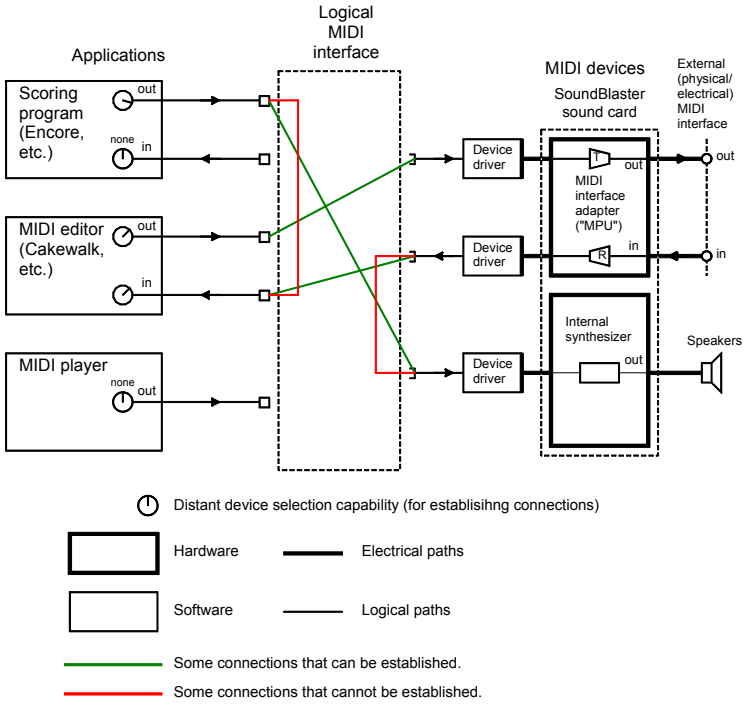

Figure 1. PC MIDI architecture

Entity gender

Because of the role of the device driver, we find that our entities have been cast into two groups, which we can think of as separated by *entity gende*r. An entity of one gender (such as a scoring program, an application) can only "link up with" an entity of the other gender (such as a synthesizer, a device). Of course, we can also only link an output on one entity with an input on another, under the concept of *port*

gender. These two gender concepts, and the limitations they provide, are distinct, but are often confused.

Figure 1 illustrates this architecture and the implications of entity gender.

On the right side of the figure, we show two illustrative familiar "hardware" MIDI entities[6] (here, "MIDI devices"), which have already been described. We also see the device drivers with which they are equipped.

The figure also shows, on the left, three illustrative MIDI-aware software applications, which have also been earlier described.

The green lines show illustrative paths for the flow of MIDI messages that can be put into effect. They all lead from an output on an entity of one gender to an input on an entity of the other gender.

The red lines show illustrative paths that cannot be put into effect because of the gender dichotomy. (I don't show unworkable attempts to link two inputs or two outputs; we should well understand this, and it would clutter the drawing.)

Who handles the hoses?

Inside our computer, how do we control how an output of one entity is linked to an input of another entity (necessarily of the "opposite entity gender")?

This is always done at the application end. For example, in the scoring program Encore, if we 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 synthesizer of interest. The result will be that the MIDI stream emitted by Encore on its MIDI output will be directed by the computer's operating system to the input of the device driver for that synthesizer.

Similarly, if I want Encore's MIDI input port (used, for example, to "capture" an incoming MIDI stream for recording as a score) to receive the MIDI stream emitted by my MIDI interface module (having arrived over a physical/electric interface from, for example, a MIDI keyboard), I open a dropdown menu for "MIDI Input". This menu lists all the

---

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

existing MIDI devices (in the sense discussed above) having outputs. I choose the MIDI interface module. The result will be that the MIDI stream emitted by the receiving portion of the device driver for the MIDI interface module will be directed by the computer's operating system to the MIDI input port of Encore.

So in all cases, the "hose handling" is directed by the application involved, and the choices are made on dialogs in that application.

The little "knob" graphics on the ports of the MIDI applications in the drawing represent this "distant port selection" process.

NOW IT GETS MORE DIFFICULT

So far, we have investigated an interesting "gender dichotomy" among MIDI entities inside the PC, but the overall organization of the system seems to take care of it.

As we seek to do some more specialized, useful, and still sensible things, we find that we are impeded. Suppose, for example, that we wanted to direct the MIDI stream generated by our Encore from a score to the input of out MIDI editor, where it can be captured as a "composition", perhaps for detailed examination of its properties, or maybe to make such subtle adjustments to the durations of certain notes. That would make sense functionally.

But we just can't do it. Since both Encore and our MIDI editor are applications, they are of the same entity gender, and the fundamental biological laws of our computer architecture do not provide for them to mate. If, on the MIDI editor, we open the "input" dialog to select a distant source, we will not find the output of Encore listed—Encore isn't a device, and only devices are listed. And of course the operating system has no direct provision for one application to send MIDI streams to another application anyway.

There is a workaround. Suppose we made a device driver that actually didn't attach to a device. It would have both an input and an output "virtual port" (just like the driver for, say, our MIDI interface module), but what comes in its input port is just transmitted verbatim through its output port. It might be best to describe it as "quasi device driver" since no "device" is actually involved.

Then we can set up the situation shown in figure 2.

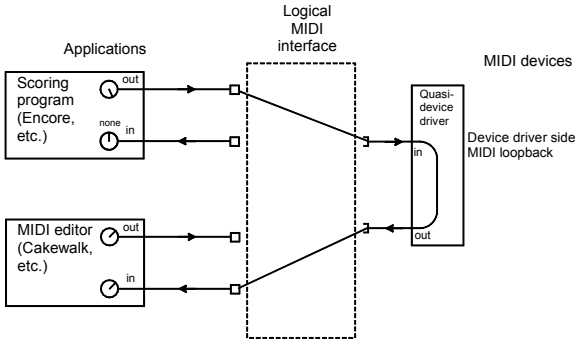

Figure 2. MIDI loopback (device side)

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

Now, let's consider another need in which we will be frustrated by the gender limitation. Suppose we would like to take a MIDI stream from the receiving branch of our MIDI interface adapter (perhaps received from an external MIDI keyboard) and just direct it to our synthesizer (so we can play a little concert on the keyboard). Again, no connection that will do this is accommodated by the basic computer architecture.

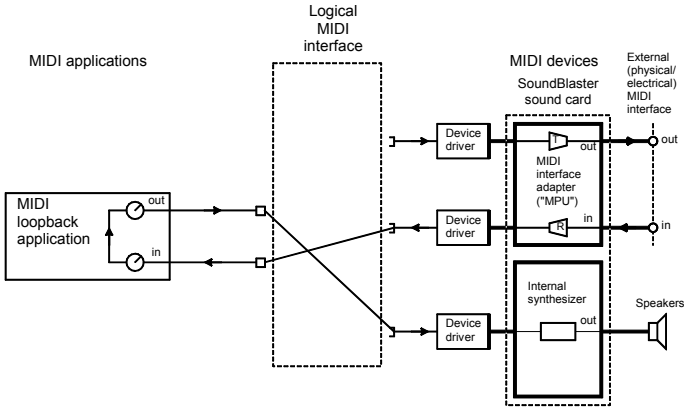

Figure 3. Application-side MIDI loopback

But again there is a workaround. Suppose we have a trivial MIDI application whose basic function is just to take a MIDI stream taken in by its input and deliver it, verbatim, through its output. We can think of it as an "application-side MIDI loopback". With it, we can establish the arrangement shown in figure 3.

Note that as always for a MIDI connection inside the PC, the "hose handling" is done by controls on the application—in this case, the "application-side MIDI loopback". Thus, in the figures, both its ports show the familiar "little knob" icon.

One widely used "application side MIDI loopback" facility is **Hubi's MIDI Cable**, published by Hubert Winkler of Vienna, a developer of various software tools for use in this general area. Winkler also publishes a device-side MIDI loopback facility, but it is out of date and evidently will not operate under, for example, the Windows XP operating system. Hubi's MIDI Cable, though, seems to work fine under Windows XP.

Hubi's MIDI Cable is (at this writing) available here (free):
http://www.geocities.com/mstella/hmidilb/hmdlpbk.html

It is included in the distribution package for Hubi's MIDI Loopback Device (the device-side loopback driver). There is an executable file and an associated DLL. No special installation is required.

Single- and multiple-client ports

Suppose we wanted both Encore and our MIDI editor to take in a MIDI stream entering the computer via our MIDI interface module? In most cases, the output port of the interface module's device driver is of the "single client" form. That is, two applications cannot contemporaneously choose to receive data from it. If we have elected this port for linkage to the MIDI input of Encore, when we later try to link it to the MIDI input of our MIDI editor, we get an error message (either now or later when one of the applications tries to "open" the nominated distant port). The error message comes from the application.

This is not an inherent "logical" limitation (such a "fan out" linkup would make perfect sense) nor is it a firm limitation of the operating system. Rather, it is a limitation imposed by a design choice for the device driver. The operating system provides for device driver output ports that can be "attached to" by more than one receiving application—said to be "multi-client" ports—but these are more complex, and often the complication is just avoided during the design by not providing that capability.

Similarly, suppose that we wanted both Encore and our MIDI editor to direct its output MIDI stream to the input of our internal synthesizer. Again, most commonly, this "fan-in" will not be allowed (the story is the same as the "fan-out" issue discussed above).

Note that even if the synthesizer input device driver were designed on a "multi-client" basis, we don't know for certain that the resulting operation would make sense, in particular when both "sending" applications were sending MIDI streams at the same time (perhaps one justification for its not being designed that way).

The MIDI Yoke device-side MIDI loopback facility has multi-client input and output ports (three "spigots" on each port for the current version). This can be useful in establishing some of the "routings" we might wish to have for various special tasks. Figure 4 shows how we might exploit this capability, in two ways.

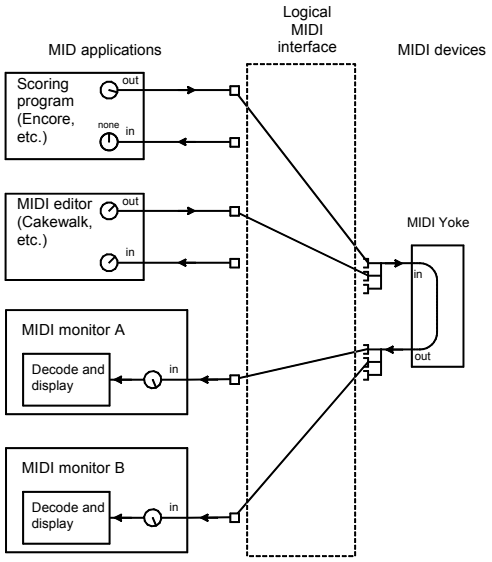

Figure 4. Exploitation of multi-client ports

A new kind of MIDI application is introduced in this example, a MIDI monitor. This is a MIDI application that receives a MIDI stream and parses, decodes, and displays the messages. It can be very handy for "debugging" both scoring programs and MIDI editors.[7]

Since this type of MIDI monitor is of the application entity gender, to send a MIDI stream from another application to it requires a device-side MIDI loopback pseudo-driver, such as MIDI Yoke.

In the example this figure illustrates we wish to be able to send a MIDI stream from our scoring program to the monitor, and later send a MIDI stream from the MIDI editor (perhaps representing the same musical

[7] The use of one kind of MIDI Monitor, **Midi Ox**, is discussed in detail in the companion article, "MIDI Monitoring in a PC", by the same author, and available at the same location as this article.

"passage") to the monitor, so that their details can be compared. We would rather not have to do any rearrangement of the connections between these two operations (as we may wish to go back and forth between them repeatedly).

By exploiting the "multi-client" nature of the input port on the loopback, we can choose it as the destination for the outputs of both the scoring program and the MIDI editor.

Then, to further enlarge our example, suppose that we have two MIDI monitors (as shown in the figure), perhaps with different analytical capabilities, and would like to direct the "specimen" MIDI streams to both. By exploiting the "multi-client" nature of the output port on the loopback, we can just choose it as the source for the inputs of both monitors.

Note that, as always, the establishment of the connections is made on the application, by "distant device" controls associated with their ports.

MIDI short circuits

The existence of these loopback tools allows the possibility that we can improvidently set up a "feedback" path (sometimes called a "MIDI short circuit"), around which a MIDI message would endlessly circulate. This is not only fruitless but can consume significant computer processing capacity. MIDI Yoke includes special capabilities to detect such a situation and interrupt the message flow.

Special capabilities

Hubi's MIDI cable provides optional "filtering" capabilities, so that (for example) certain classes of MIDI messages, or messages tagged for a certain MIDI channel, are blocked. The facility can also transform certain messages into other messages, or redirect all messages tagged for a certain MIDI channel to another channel. These capabilities can be useful in performing special tasks. The details are beyond the scope of this article.

Often, insinuating this tool into "normal" a message path, so it can perform such interventions, is frustrated by entity gender considerations, since its input and output ports are of the same entity gender (application). (Imagine a garden hose in-line shut-off valve with a male hose fitting on each end!) Thus, we may often have to employ a loopback module of the other gender (a device-side loopback) with it, just to make the gender matters "come out even".

Loopback via "conventional" applications

We saw in some of the figures, and briefly noted, that the typical scoring program and the typical MIDI editor will have input, as well as output, ports. As mentioned, these are typically intended to receive MIDI streams from, for example, an external keyboard, a way for the composer to quickly enter the notes of the musical passage to be stored for further refinement.

Almost always, these applications provide for a "MIDI through" capability, through which the MIDI message stream arriving through the input port may be (in addition to being recorded) sent out the output port. One motive for this is so that we can send the stream to our internal synthesizer so that we can hear what we are playing.

We can then see that a "trivial" use of such an application is just as an "application side loopback" for various situations.

Although not illustrated here, typical MIDI monitor applications also provide the "MIDI through" capability, and can also intervene in the MIDI stream being "passed through", just as we discussed for Hubi's MIDI cable. As with that situation, since both ports of the monitor are of the application entity gender, we may well have to employ a device-side MIDI loopback to complete the path that we need.

#