MIDI—The Musical Instrument Digital Interface

Douglas A. Kerr

Issue 7
March 8, 2009

ABSTRACT

The *Musical Instrument Digital Interface* (MIDI) can link together, electrically, various entities involved in the generation, storage, or execution of musical performances. This interface is the centerpiece of an entire complex paradigm of note-oriented digital representation of musical performances. This paper reviews the basic concepts of this interface, of the paradigm it spawned, and of the related matter of the *Standard MIDI File*, a format for storing in a computer file the MIDI instructions for executing a musical performance. It also discusses the related concepts of MIDI-oriented musical notation software, the MIDI sequencer, and MIDI interface hardware. The "MIDI language", and its repertoire of MIDI messages, is discussed, with full detail, in an appendix.

INTRODUCTION

The musical instrument digital interface (MIDI) is a standard physical, electrical, logical, and syntactic interface which, in its most obvious application, links together a *MIDI controller*[1] and one or more *MIDI sound modules* in order to allow the automated performance of music. It is the centerpiece of an extensive and complex paradigm of note-oriented digital representation of musical performances.

MIDI keyboard controller

There are two classical units of the MIDI controller category. One is a MIDI interface[2]-equipped musical keyboard (meaning just the keyboard proper, not the entire "instrument" comprising both a keyboard and a music synthesizer, often nevertheless popularly called a "keyboard").

MIDI sequencer

The second classical kind of controller is the *MIDI sequencer*, an entity that can automatically emit over a MIDI interface the stream of

---

[1] Careful—the term *controller* is also used in a wholly different way we will encounter shortly.

[2] Although the last "I" of MIDI stands for "interface", for syntactic ease we generally speak, tautologically, of a "MIDI interface".

instructions for a musical performance (the MIDI sequence) from stored information. A sequencer may be a dedicated piece of hardware, typically able to read from floppy disk a file describing the performance, or it may be implemented by application software running on a general-purpose computer (PC or Mac, for example) which is equipped with a MIDI interface adapter.

There are in fact three principal types of entity that precisely meet this definition, but by industry custom, only one is called a "MIDI synthesizer". I'll discuss this later under "MIDI Software.

MIDI sound module

A MIDI *sound module* may be an actual musical instrument, equipped with electrical actuators for "playing" it, or it may be a *synthesizer*, an electronic unit capable of making musical sounds, often emulating the sound quality (*timbre*[3]) of one or more traditional instruments, and likely also having a repertoire of unique timbres not matching those of traditional instruments.

Although we typically characterize the role of the MIDI interface as described above, with the "performance" being transmitted from the controller to the sound module, it may also be used in such settings as transmitting a performance from a keyboard to a sequencer so the performance may be stored for later automatic rendition.

CAVEAT

The syntactic realm that revolves around the MIDI interface is extensive, complex, and rife with peculiarities. Overall, it is well beyond the scope of this article. Here, I hope to give enough of an overview to equip the reader to deal with MIDI-oriented matters.

The details given in this article for the most part relate to the "original" MIDI specification. An updated specification. "MIDI Level 2", was introduced in 1999. It makes certain changes which will not be treated here.

THE PHYSICAL/ELECTRICAL INTERFACE

Introduction

The information that follows is for the original physical/electrical interface. In modern times, analogous specifications have been developed for transmitting MIDI message streams over other types of physical/electrical interface, such as USB, Firewire, and so forth.

---

[3] Pronounced as in French: "tam'-bruh".

These are beyond the scope of this article. The message structure and syntactic implications described here do not vary over the different interfaces.

Basic electrical interface

The MIDI interface is potentially a full-duplex data transfer interface, but the two directions of transmission (where applicable) are carried by electrically (and physically) separate circuits. Each "direction" is carried by a two-conductor unipolar current loop (like a neutral telegraph loop). The loop current is nominally 5 mA.[4] The sending end is active (i.e., supplies the voltage for the loop current). The sending circuit is not typically balanced with respect to ground. The receiving circuit must be floating, and its implementation often involves an opto-electronic isolator.

The idle condition is "marking" (current flowing), which represents logical "1". Current not flowing is the "spacing" condition, which represents logical "0".

The physical interface

The standard physical presentation of the MIDI interface is on a 5-pin (180°) "DIN" connector (identical to that used for personal computer keyboards a few years ago), with female pins, separate connectors being provided for inputs and outputs when both are present. Normally, the implementation is as a "chassis-mounted" jack.

The same pins (pins 4 and 5) are used for the actual data circuit at both input and output connectors; pin 4 is the positive side. Pin 2 is grounded at output jacks only (to provide for a shield ground for the interconnecting cables).

Interconnecting equipment items is done with "MIDI cables", which have identical male-pin plugs on both ends. One end is plugged into an output jack, and the other end into an input jack (ordinarily on another unit). If bidirectional transmission is involved, two such cables must be used.

The cable wiring is "pin-to-pin" for pins 4 and 5. Pin 2 connects to the cable shield at both ends.

The input-output "duality" can make it tricky to label cables in an installation for identification.

---

[4] The actual current may be as much as 7.5 mA, depending on implementation and connection details.

Transmission format

Transmission is on an asynchronous ("start-stop") basis at a rate of 31.25 kilobauds. The idle condition is marking ("1"). The character (byte) format is one start bit (spacing, or "0"), 8 data bits, and one stop bit[5] ("marking", or "1"). Of course, if the characters are not being sent at maximum possible rate, the stop bit of a character will be prolonged (becoming a period of the "idle" condition) until the next character commences (as for any asynchronous serial transmission).

The lowest-order data bit is sent first (as for the serial transmission of ASCII characters).

Message structure

MIDI information is sent in the form of MIDI messages. With the exception of one specialized message type, MIDI messages consist of one to three consecutive characters (bytes). The first, called the *status byte*, indicates the type of the message (in effect, the message "verb") and also (for most messages) the channel number for the message. The second and third bytes, when present, are *data bytes*, and carry the parameter(s) of the message. The number of data bytes (0-2) is constant for any given message type[6] (and thus will be known as soon as the status byte is interpreted).

The status byte always has a most significant bit value of "1", while the data bytes (if present) always have a most significant bit value of "0". This allows the receiver to unambiguously synchronize with the message format. Thus there are actually only seven true data bits in a byte, and so, in the data bytes, parameters as coded usually have an available range of 0-127. In the case of two particular message types, the data bits of the two data bytes together represent a single 14-bit value.

Further details of message structure and syntax will come later, with even further detail given in Appendix A.

---

[5] In rigorous terminology, we speak of start and stop *elements* rather than *bits*, since these do not actually carry information. This subtlety is largely ignored in modern practice, as I did here.

[6] Except for that one specialized type, which has a "variable length" structure.

OPERATIONAL CONCEPTS

MIDI channels

Most MIDI messages are tagged with a *channel number*, which can range from 1 to 16 (in "human" notation; the actual coding has a range of 0-15, carried in four bits). This allows several separate streams of MIDI messages to be carried over a single electrical interface, essentially traveling over "virtual channels". This can serve at least two purposes. Firstly, it allows several physically or logically distinct MIDI devices to operate on a single electrical interface path (usually with a "daisy chain" arrangement), eliminating the need for multiple output ports on the MIDI controller.

Secondly, many MIDI sound modules are able to emulate, at the same time, multiple musical instruments with different timbres. The channel tag system allows the note messages destined for these different virtual "instruments", traveling across the same interface, to be distinguished at their destinations.

Polyphony and multitimbrality

A musical instrument (including a MIDI sound module) which can only sound one note at a time (as for an actual trumpet) is said to be *monophonic*. An instrument which can sound more than one note at a time (such as an actual piano) is said to be *polyphonic*, or to have the property of *polyphony*.

A sound module which can, contemporaneously, act as several different instruments, each with a different timbre, is said to be *multitimbral*, or to have the property of *multitimbrality*.

Programs, patches, and instruments

As mentioned above, many synthesizers may emulate the timbres of different traditional instruments, as well as, typically, some timbres unique to "electronic music".

The MIDI controller may command the sound module to put into effect a particular timbre from its repertoire with a *program change message*.

The actual provisions in the sound module which cause the different timbres to be sounded are often called *patches*. This term goes back to the days of analog synthesizers, in which a particular timbre was set up by making connections among various oscillators, modulators, and filters using small plug-in cords (called "patch cords" by reference to the cords used to interconnect audio apparatus) on a plugboard. A particular arrangement (resulting in a particular timbre) therefore came to be known as a "patch", a term which still is used in the equivalent sense for digital synthesizers.

A program change message with a certain program number will cause the sound module (synthesizer) to put into effect a certain patch. But the assignment of program numbers to particular patches may be arbitrary in any particular device.

The different patches a synthesizer can adopt are often spoken of as "instruments".[7]

A multitimbral synthesizer can adopt, contemporaneously, more than one timbre, and thus can play several "parts" of a musical arrangement at the same time. The different aspects of the synthesizer that do so are in fact often spoken of as **its** different "parts".

In transmitting an actual musical performance over a MIDI interface, the messages destined for the different "parts" are sent on different MIDI channels (that is, tagged with different channel numbers), generally assigned arbitrarily. At the beginning of the performance, individual program change messages are issued on each channel to designate the timbre of the "instrument" which is to be utilized for playing the notes subsequently received on that channel.

The General MIDI Specification

Traditionally, the repertoire of available timbres ("patches") in a synthesizer, and the program numbers (in the program change message) which put each into effect, were proprietary to the synthesizer manufacturer. There is, however, an industry standard repertoire of named timbres ("instruments") with associated program numbers, known as the "General MIDI Specification". Any synthesizer which follows, or can be configured to follow, this specification can play any MIDI performance predicated on the General MIDI (GM) context with the proper timbres (however it is best able to do so).

Comparable standard arrangements developed by specific synthesizer manufacturer are also used. An important one is the Roland "General Standard", or GS, which is really an extension and elaboration of the General MIDI standard. Another is the Yamaha "XG" standard.

An industry standard extension and expansion of the MIDI/General MIDI standards, "MIDI Level 2", was introduced in 1999.

---

[7] But sometimes an entire synthesizer is also referred to as an "instrument".

Percussion

With respect to their handling in a MIDI context, percussion instruments can be divided into two categories. *Chromatic* (or *melodic*) percussion instruments sound notes of a scale, with distinct pitches, as in the case of a marimba, glockenspiel, or a set of tympani (of course, an actual suite of tympani can only sound certain notes at a given time). These are treated under MIDI just as any other instrument. The General MIDI Specification instrument repertoire includes a number of chromatic percussion instruments.

Non-chromatic percussion instruments do not sound various notes, and usually do not have a pitch in the usual sense. Typical examples are snare drum, bass (kick) drum, triangle, cymbal(s) of some particular flavor, wood block, etc. Thus, to tie up an entire "program" number and a separate MIDI channel for each non-chromatic percussion instrument would be a waste of code capabilities.

Instead, a whole collection of non-chromatic percussion instruments are gathered together to form a single MIDI "instrument". Different note numbers cause the different percussion instruments to be sounded. For example, note 38 might cause the acoustic snare drum to strike, while 36 might make the bass drum strike, and 49 the crash cymbal.

A particular collection of synthesized non-chromatic MIDI percussion instruments (or "traps" as they are often called), treated as one MIDI instrument, is often called a "drum kit".

As with the regular instrument patches, the traps which are included in the drum kit and the notes assigned to each had traditionally followed a manufacturer's proprietary arrangement. The General MIDI Specification, however, prescribes a standard drum kit with a standard assignment of notes to the different traps.

Further, the General MIDI Specification provides that MIDI channel 10 should be used for messages to the drum kit. (MIDI Level 2 provides than channel 11 should be used for a second drum kit to expand the number of traps accommodated.)

MIDI MESSAGES

In this section we discuss some important MIDI messages types, those most needed to "make music".

Note messages

MIDI **note messages** tell the sound module to begin sounding a particular note, or to stop sounding it. These messages are sent in real time, and ideally are executed immediately by the sound module.

The Note On message carries a channel tag and two parameters (in its two data bytes):

- The *MIDI note number*. This identifies the note as an integer indicating its pitch in the traditional (Western) musical scale; each successive higher integer represents a pitch a half-step higher (that is, corresponding to successive keys on a piano keyboard, both white and black keys being counted). This value is in the first data byte of the message.

  Note number value 60 represents middle C ("C5" [8]). The note transmitted is always the note to be sounded, even in the case of "transposing" instruments. [9]

- The *note velocity*. On a piano, the velocity with which a key is "struck" affects the loudness (and to some extent, the timbre, especially at the beginning) of the note sounded. Thus, for a piano "patch", the velocity parameter of a note message affects the volume with which the note is sounded and the timbre as well. This value is in the second data byte of the message.

  For most instruments other than the piano, the velocity parameter still signifies the loudness at which the note should be sounded (and again, perhaps the subtleties of its timbre, if that would change as the prototype instrument were played louder or softer.) However, in the case of a note message intended for a synthesizer emulating a pipe organ, the velocity parameter is usually ignored, reflecting that the loudness of individual notes played on a pipe organ cannot ordinarily be varied from the keyboard.

The definition of the MIDI language provides a distinct Note Off message (a different message type), used to command that the sounding of a note cease. This message contains the channel tag and

---

[8] This notation means "C in octave 5". Unfortunately, in some contexts middle C is designated as "C3" or "C4". These all refer to the same note, and its MIDI note number is still 60.

[9] The note shown on a trumpet score as "middle C" when played by a "B♭" trumpet has the actual pitch A# (B♭). To make that note sound on a synthesizer, regardless of the patch in effect, we must send a message with note number 58 (B♭). Scoring programs will generally let us make this adjustment when the score is "played".

the note number (which identifies the note, now sounding, that is to be stopped); the note velocity parameter is usually a dummy (although in theory it can indicate different "speeds of release" of the piano key). It is common to use an arbitrary value of 64 for this dummy parameter if it has no actual significance.

However, modern practice sometimes handles the Note Off operation in a different way altogether. The Note On message is used, but with a velocity parameter of zero. This is taken to mean "Note Off". This peculiar convention is intended to facilitate the employment of a type of "message shorthand" called the *running status* convention (discussed shortly).

Program change message

The MIDI controller may command the sound module to put into effect a particular "patch" from its repertoire with a **program change message**. This carries a parameter with value from 0 to 127, which is usually deemed to represent a program number from 1-128.[10] The patch implied by the specified program number will be applicable (until further change) to all note messages having the same channel tag as that of the program change message (which we can think of as describing notes to be played by a particular "instrument", a particular "part" of the synthesizer). As mentioned above, the association of program numbers with patches may be arbitrary, or may follow the General MIDI standard mentioned earlier.

Control change messages

Although the entire device which emits a sequence of MIDI messages is called, generically, a *MIDI controller*, the term *controlle*r has another totally different meaning. For example, the sustain pedal on a piano is considered a *controller*, or a master volume control for the instrument being played.

Control change messages (often called **controller messages**) reflect the states of these controllers. One subclass (a *switch* controller message) is used for controllers which only have two states (such as Sustain Pedal: on/off). They carry a Boolean (two-state) parameter. Another subclass is the *continuous* controller message*,* which has a numeric parameter (such as channel master volume level).[11]

---

[10] But sometimes the program numbers are considered to run from 0-127, a source of much confusion!

[11] The term *continuous* is not rigorously accurate, in that the value is not truly continuous but discrete, usually quantized to no more than either 128 or 16,384 possible values. The intended distinction in the term is with a two-state controller.

The "pitch bend" wheel on many electronic keyboards, which allows the performer to slide the pitch of the notes being played up or down in pitch, is in fact a type of continuous controller. Its setting, however, is not sent in a specific continuous controller message as such, but rather in a unique *pitch wheel* message, with 16,384 distinct values (its two data bytes collectively give a 14-bit parameter).

Other messages

There are many other MIDI messages. Most of the MIDI messages are listed in Appendix A (and the ones not listed individually are identified by class), which also gives further detail that we see just above as to their format, coding, and syntax.

Running status

Often we will need to send a long sequence of messages with the same "verb" (status byte value). To save transmission time, the status byte can be omitted from all such messages after the first. The receiver is alerted to the use of this by the fact that, after what should be the end of the first message (its length is indicated by the message type, as given in the status byte), another data byte is found (recognized by the value of its most-significant bit). Since there will be the same number of data bytes in all messages of the "series" (they all have the same message type, given by the status byte of the first message), the data bytes of the subsequent messages can readily be "parsed out" and interpreted.

This technique is called *running status*, and can be thought of as a type of transmission shorthand. Its advantage is in reducing the number of bytes to be sent for a musical sequence. It is only workable for sequences of messages pertaining to a single channel, since the channel number is embedded in the status byte.

We can exploit this shorthand for a sequence of note messages by following the convention in which the degenerate case of the Note On message (with velocity = 0) is used to end a note rather than the distinct Note Off message. Thus, both Note On messages and this special form of Note Off message have the same status byte ("Note On"), and so the whole sequence can be handled under the running status protocol. In fact, it is for the note messages that the transmission economy of the running status convention is most potentially profitable, since these normally constitute the majority of the messages.

Bank selection

The basic concept of program change allows us to, at any given time, select among up to 128 different patches to be used to render the notes that will arrive over a certain MIDI channel.

As the capability (and sophistication of use) of synthesizers advanced, it was found that this range was insufficient for the enlarged repertoire of patches we might wish to implement in a synthesizer.

Thus, to extend the patch selection "code space", the concept of *bank selection* was introduced. Basically, this provided a "paged catalog" of patches. The "page" to be used (called a *bank*) was indicated by use of a Bank Select message (one of the group of "control change messages, the one identified as "CC0"). It could distinguish among up to 128 banks. That having been done, the normal Program Change message would subsequently designate the particular patch (from up to 128) in that bank.

While it might seem that this gigantic range of bank/program identifiers (each potentially indicating a particular patch)—16,384 altogether—should more than well satisfy emerging synthesizer capabilities, the range was often used in an inefficient way.

For example, a synthesizer manufacturer might have program 1 on bank "0" (the "default bank") indicate the patch for a "regular" grand piano. Program 1 on bank 1 might indicate the patch for "grand piano—variant 2", and program 1 on bank 2 "grand piano—variant 3".

This gave us a "very sparse" code space. If the most variants we had for any kind of instrument was 7, and only one instrument had that many variants, then only 7 banks would actually ever be used, and the last of them would only have one patch assigned (program 1: "grand piano—variant 7").

So there was agitation for an even larger repertoire—for the ability to select from more than 128 banks.

This was achieved by increasing the size of the parameter that indicated the bank from 7 bits to 14 bits. This was done by defining another control change family message (CC32) to contribute 7 more bits to the bank identity (7 bits being delivered by CC0).

Now, these two CC messages (CC0 and CC32) were redesignated "bank select MSB" and "bank select LSB" respectively ("most significant byte" and "least significant byte"). These terms are, of course, technical misnomers, since each message does not contribute 8 bits (a byte, or "octet") to the overall bank number but only 7 bits (a "septet"). Thus, "most significant septet" and "least significant

septet" would have been more apt, but in a field where the workers are mostly musicians, you can imagine the confusion that could cause!

Now, why were the "most" and "least" assignments made this way? Well, if we start with a coding system with a certain range, and desire to increase the range by adding more digits, there are two possible outlooks. If we want the number to actually cover a greater range, we add digits on the most significant end (as if we increase the "house number" field in a data base from four decimal numbers to five, to accommodate house numbers like "16606").

If instead, we are interested in a greater "refinement" of the indication, we add digits on the least significant end (as when we increase the "inventory value" field in a system from seven decimal digits to nine so we can show values to the nearest cent).

Now one could argue, tortuously, that in seeking to increase the number of banks that we can distinguish from 128 to 16,384 we were looking for "greater refinement". But it is most reasonable to say that we just want "more numbers". (After all, a bank number isn't a quantity like temperature—it is a "counting number").

This, it would have seemed sensible to consider that the added bank selection message (CC32) would add **higher-order** bits to the 7-bit number carried by the original message, CC0.

But when this was being done, the range of almost all the continuous controller "commands" was also increased from 7 to 14 bits by the assignment of complementary messages, and in almost all cases other than bank selection, the objective was to increase the "fineness" of control. Thus, in all those other cases, it was most sensible to consider the added message as contributing lower, not higher, order bits. For what some people thought was consistency, this outlook was carried to the bank selection message pair.

Accordingly, CC0 (the original message) became "bank selection MSB" and CC32 (the supplementary message) became "bank selection LSB".

Now, not all synthesizer manufacturers actually implemented the potential for such a gigantic bank repertoire. And thus some of their machines would only respond to one of the bank select messages (which would allow selection of up to 128 banks—quite enough for these machines).

But which one? Some manufacturers said, "Well, if we are not going to need the entire range, and thus will not need the entire 14-bit

number, only the least significant 7 bits", so their machines respond to CC32.

Other manufacturers (notably Roland) in effect said, "before we had CC32, there was only CC0, and so of course that is the most basic part of the number (in effect, declaring the bits of CC0 to be the 'least significant', notwithstanding the designation in the specification), and having these limited machines respond to that message would give the greatest compatibility with existing practice". Thus they made their basic machines respond only to CC0.

The notation program Encore (in versions prior to version 5) only allows the user to arrange to have one bank select message sent out, with its value chosen by a 0-127 setting in a dialog. But, to deal with the situation described above, one can check a box marked "Roland GS", which makes the message being sent CC0 rather than CC32.[12]

THE MIDI FILE FORMAT

Introduction

An entire musical performance, intended for transmission over a MIDI interface to one or more MIDI sound modules, may be stored as a computer data file in what is known as the *MIDI file format* (MFF) or *standard MIDI file format* (SMF). These files today usually have filetype extension .MID.

Typically, a MIDI sequencer will read the MIDI file and, from it, generate in real time a sequence of MIDI messages that will cause a synthesizer to render the musical performance described by the file. The entire performance may be spoken of as a *MIDI sequence*.

MIDI events

The most common entities stored in a .MID file are called *MIDI events*, and correspond to MIDI messages. The most widely used is the **note event**. This contains the same parameters as the MIDI Note On or Note Off message (message type, channel number, note number, and note velocity).

For each event (of whatever type), the record also includes a time parameter ("delta time"), which gives the time at which the event is to take place (that is, the time the event message is to be sent over the interface), measured from the instant of the previous event. The time is commonly given in *MIDI ticks*, a certain fraction of a quarter note

---

[12] As of Version 5, Encore allows the user to set both the "MSB" and "LSB" values for bank selection, and both messages are sent out when appropriate.

(the particular fraction being declared at the beginning of the file).[13] Thus, the actual interpretation of the delta time parameters at the time of performance, as "real time" intervals, is dependent on the tempo in force at the time (typically set by the "operator" on the sequencer's control panel, but perhaps directed by the file).

Tracks

In one style of standard MIDI file ("Format 1") events in a MIDI file are gathered together into groupings called *tracks*. Typically, the note events in a track represent a particular instrument's part of a musical arrangement and/or a particular staff on the musical score for the arrangement.

The track to which a particular note event is assigned has no effect on the transmitted MIDI message (and does not show up in the note messages). The assignment of note events in the MIDI file to different tracks is solely for convenience in managing the musical "arrangement" in such contexts as the use of "musical notation" or "MIDI sequencer" software (which we will review later).

Of course, if the notes in different tracks are intended for physically different sound modules, or for different instruments being implemented by a module, their MIDI note messages must in fact carry distinct channel tags. Thus, often there will be a one-to-one correspondence in the MIDI file between the track assignment of an note event and the assigned channel number. But notes assigned to different tracks (such as two tracks associated with the treble and bass staves of the piano part) may result in MIDI note messages with the same channel number if they are to be played on the same "instrument" (as is usually the case for the piano part).

Interestingly, in the Standard MIDI File format, all the events assigned to a single track are recorded sequentially, followed by all the events in the next track, and so forth. This means that a sequencer, preparing to emit the sequence described by the file, must read into memory the entire file, and then "collate" the different events, after they have acquired channel tags,[14] in time order.

---

[13] This scheme, which describes time in terms of a note value, is called *metrical time*. An alternative uses "clock time", defined in a specialized system used in connection with motion picture and TV timing, called the *SMPTE time code system*.

[14] The user may set the channel tag to be used for the notes in each track on the sequencer controls, or that decision may have been made earlier, and the channel to be used will have been recorded in the track information (see the discussion under meta-events, just below). Thee is in general no opportunity to have different notes that are part of the same track emitted with different channel tags.

Meta-events

The MIDI file may also contain meta-events, which do not cause the emission of MIDI messages. Rather, these are directions to the sequencer that will be emitting a MIDI sequence. Among the types of meta-events are:

- Instructions as to what tempo to follow when emitting the MIDI sequence.
- Instructions as to what MIDI channel tag should be applied to the note events from each track.
- Lyric meta-events, which record "lyric particles" (typically, one-syllable words or individual syllables of multi-syllable words).

MIDI SOFTWARE

Three types of software packages for the PC are most directly pertinent to our interest in MIDI, *music notation* software, *MIDI sequencer* software, and *MIDI player* software. As will become apparent, there may be considerable overlap of their functions.

Music notation software

Music notation software (sometimes called "scoring software", or a "scorewriter") basically does for musical scores what word processors do for mostly-textual documents and what CAD programs do for engineering drawings.

Modern programs of this genre allow the preparation of music scores ("sheet music") following the full range of musical notation. They typically include provisions for showing notes and rests of different value, key signatures, time signatures, tempo changes, dynamic markings, repeat notation, lyrics (words), titles, copyright notices, and the like.[15]

In addition, most notation programs maintain in memory a data set, conceptually like a MIDI file, from which a MIDI sequence describing a "performance" of the score can be generated and directed through a MIDI interface to a MIDI sound module, thus rendering the music represented by the score. These programs usually can also, on command, generate a MIDI file carrying this performance.

---

[15] This office uses Encore by GVox

For both of these operations, the channel number and patch (in terms of a program number) for each staff of the score may be set by the user.

The score may be "composed" with a number of techniques. The individual notes and rests may be entered "on the score" with a mouse. These elements may also be entered by manipulation of the QWERTY keyboard.

Additionally, an actual MIDI-oriented keyboard, connected to the computer through a MIDI interface adapter, may be used to send into an input port of the notation software. The notes "received" will be deposited on the score.

Many of these notation programs are also able to import a MIDI file and, from its contents, deduce the nature of the score which might have directed its generation, and display this score on the screen.

MIDI sequencer software

A MIDI sequencer, like a notation program, allows us to compose or modify a note-oriented musical performance but with a paradigm that emphasizes the representation of the performance as MIDI events rather than as notes on a score (as in a notation program).[16]

Many MIDI sequencers can display the stored performance in several ways. One way is to deduce (from the defining sequence of MIDI events) the *score* which would have originally described the performance and display it. Another common mode is the *piano roll* display, which has a horizontal "track" for each note value, with dark bands showing when that note is sounding. This is especially useful for the many composers and performers who do not read nor write regular music notation, or to allow direct visualization of subtleties of note timing.

Another common display is the *event list*. This is sort of like an assembly language source listing. Each MIDI event is listed, with all its parameters. This is the most direct reflection of the actual performance as stored in the sequencer. In most cases, the sounding of a note is listed as a single event (even though it is executed by two messages, Note On and Note Off), with its duration indicated as though it were a parameter of the event.

Often, the events in the event list may be edited directly. Thus, a particular note, shown on the score as an eighth note, can be made

---

[16] This office uses Cakewalk Home Studio by Twelve Tone Systems

just a little longer (as a human performer might do to achieve a certain rhythmic effect), or may be given a different velocity parameter so the single note is a little louder than its neighbors.

Similarly, in many programs, the note values seen in the piano roll view may be changed in starting and ending times with the mouse.

Changes entered on one view are then reflected on the other views.

Although as mentioned, many sequencer programs can display a performance in conventional score form, they typically do not have the range of features offered by notation programs for flexibly controlling the format and appearance of the score and the various collateral notations used on a typical published score.

MIDI player software

A MIDI player is an entity that takes a stored musical performance and emits a sequence of MIDI messages (perhaps to a synthesizer) that will cause the performance to be executed.

And of course, both notation software and MIDI sequencers perform the MIDI player function when "playing" the performance.

MIDI sequencer—the broader meaning

Note that all three of the program types described above seem to meet the definition we saw much earlier of a MIDI sequencer, yet only one is, by custom, usually called that.[17] Notwithstanding that custom, in this article I will use "MIDI sequencer" to mean any entity that can emit a MIDI sequence from a stored representation. (Otherwise, I would have to invent a new word for that—perhaps "MIDI sequence emitter"—and wouldn't that be silly).

COMPUTER HARDWARE

The sound board

MIDI interface capability for a modern PC is usually implemented by a separate *sound board* or an equivalent functionality on the computer's *mother board* itself. This typically combines a number of functionalities, including:

a) An encoder-decoder (codec) for recording and reproducing sounds in waveform-coded digital form. These are often found stored in

---

[17] Defenders of that custom generally say (a) it isn't really a sequencer unless the user can compose the sequence in it, and (b) composing a sequence by working in a score paradigm workspace doesn't count, as it isn't MIDI-enough.

WAV files. The encoding and decoding is typically by way of the pulse code modulation (PCM) technique, similar to the way speech is encoded for transmission and switching in the telephone network.

b) One or two polyphonic, multitimbral synthesizers. Two types have been used. The earlier type uses a synthesis technique called FM synthesis, which involves the frequency modulation of one audio frequency sine wave by another, allowing the creation of a wide range of waveforms (and thus timbres). The more modern synthesizer type utilizes *wavetable synthesis* or a related technique. Here, actual specimens of the sound of the instrument being emulated are stored in PCM form in the synthesizer. These waveform specimens are then scaled in frequency for the note being requested. The result is extremely realistic simulation of traditional instrument timbres.[18] These synthesizers are driven by MIDI message streams.

c) An external MIDI interface adapter. This provides for the connection of the computer to traditional, physical MIDI devices such as self-contained synthesizers, physical instruments arranged to be electrically "played", or MIDI keyboards. In most cases, the sound board itself (or the mother board equivalent) does not contain the full electrics for the MIDI interface. Instead, the MIDI message streams, in standard MIDI logical format, travel through the connector (not a standard MIDI connector) as TTL-level electrical signals. To convert these to the unipolar current loop format of the MIDI interface proper, an external adapter is needed. This is often built into a "MIDI interface adapter cable" provided by the board manufacturer (or others). It's done this way so that, considering only a small fraction of sound board users will actually use the external MIDI interface, the cost of the board itself can be reduced, as the loop sending and receiving circuitry itself involves special discrete analog components.

A popular family of sound boards, which have set *de facto* standards for the internal interface with the music application software, is the Sound Blaster series, made by Creative Labs, Inc.

---

[18] Timbre involves more than just waveform. The attack and decay characteristics of the notes also figure into timbre. Both "FM" and "wavetable" synthesizer approaches provide for specific attack and decay characteristics for the individual "patches".

What is "MPU-401"

The earliest commonly-used external MIDI interface adapter for the PC was the Roland MPU-401 ("MIDI processing unit"), packaged in the form of an ISA-bus expansion board. This unit set *de facto* standards for the internal "logical" interface between the music application software and the MIDI interface port itself.

As a result, when we select the destination for the output from a music notation package or MIDI sequencer, we may find a choice identified as, or making reference to, "MPU-401". This merely means the external MIDI interface adapter unit.

The logical interface

MIDI entities inside a computer are not tied together over the traditional MIDI physical/electric interface, but rather over a "logical MIDI interface, which is a particular form on the interface used inside the computer between applications and I/O devices. Such entities as the synthesizer and external MIDI interface adapter portions of a sound board are provided with device drivers. Applications such as scoring software, MIDI sequencers, and MIDI players then link to these devices in much the same way that a word processor would link to a printer through its device driver.

A corollary of the use of the "device driver" interface as the basis for this logical interface is that the interconnected entities are cast into two "genders", those that are like *I/O devices* and those that are *applications*. We cannot, using the normal computer capabilities, arrange for the travel of MIDI messages between two entities of the same gender. Thus there is no direct way to arrange for the stream of MIDI messages emitted by, say, a scoring program as it "plays" the score, to be directed into the receiving port of a MIDI sequencer.

There are available special software accessories that can serve to overcome this limitation of the "MIDI plumbing" in our computers. Discussion of these is beyond the scope of this article.

MULTI-SEQUENCER PERFORMANCES

For various reasons, the resources used to produce a musical performance may be under the control of more than one sequencer. An important example is the inclusion of a "drum machine", a highly specialized sequencer closely coupled to a synthesizer, devoted to percussion parts. The entire suite of percussion parts is stored in the drum machine sequencer, while another sequencer has all the other parts of the performance.

Just as with separate human performers, it is necessary to properly synchronize the operation of multiple sequencers. One sequencer (typically the one managing the non-percussion parts) will play the role of "conductor". It will instruct the subordinate sequencers when to start and stop, indicate from which point in the score they should start, and provide a "beat" to keep the process moving forward in synchronism.

Administering this modality is the function of a special set of MIDI messages, one of which is a MIDI System Common Message and the others of which are MIDI Real Time messages The important ones are:

- MIDI Sync (Real Time)—this message is sent at regular intervals (in terms related to "beats", not clock time). Subordinate sequencers pace their output based on this received beat stream.

- MIDI Start (Real Time)—this message tells the subordinate sequencer(s) to commence play (from the beginning of the score) upon the arrival of the next MIDI Clock message.

- MIDI Stop (Real Time)—this message tells the subordinate sequencer(s) to commence play (from the beginning of the score) upon the arrival of the next MIDI Sync message. When play stops, a *song position pointer* in the subordinate synthesizer is set to the next clock position after the last clock position played, in anticipation of a possible restart with the MIDI Continue message (see next).

- MIDI Continue (Real Time)—this message tells the subordinate sequencer(s) to commence play (from the point indicated by its song position pointer) upon the arrival of the next MIDI Clock message.

- MIDI Song Position Pointer (System Common)—this message forces the song position pointer of the subordinate sequencer to the value carried by the parameter of the message, in anticipation of start of play from that point by way of a MIDI Continue message.

The MIDI Real Time messages have no parameter. The MIDI Song Position Pointer message carries a 14-bit parameter.

***KARAOKE* WORK**

Karaoke[19] refers to an entertainment, popular in bars and private parties, in which a song is played from a recording while the lyrics appear on a screen, usually with a prompt (such as the "bouncing ball" prompt that was used decades ago for a similar entertainment in motion picture theaters). The revelers show their vocal skills in this context.

It is common for these songs to be recorded as Standard MIDI Files, sometimes with some special wrinkles.

There are two conventions for embedding the words in the file. In one, the MIDI Lyrics meta-event is used (essentially as originally intended). In other cases, the MIDI Text meta-event is used. This provides for the embedment of text for various reasons (not originally intended to be for lyrics to the musical song), and has been hijacked here for lyrics use. (This may have been precipitated by some patent issue.)

There are many nice MIDI players that have a Karaoke feature, often allowing the onscreen display of lyrics recorded either as MIDI Lyrics meta-events or as MIDI Text meta-events. They are often used in contexts that are socially different from the Karaoke context, such as providing for church choristers to learn their parts of hymns.

In connection with the use of the MIDI Lyrics meta-event in a Karaoke context, the convention arose that a Lyrics meta-event at "time zero" would be used to carry identifying information about the file, not the first lyric particle, and thus should not be displayed on-screen.

Workers planning to use the MIDI Text meta-event for Karaoke, choir practice, or whatever, need make the first Lyric meta-event, if associated with a note at "time zero" (typically shown in MIDI time notation as 1:01:000—"first measure, first beat plus 0 ticks") have a time tag of 1:01:001.

Sometimes, Standard MIDI files set up for Karaoke work are given the .KAR filetype extension.

CONCLUSION

The MIDI interface and its related protocols, file formats, and software packages have brought to composer and performer, musician and programmer, amateur and professional, powerful tools to greatly enhance their productivity in utilizing, improving, and enjoying their musical talents and inclinations.

---

[19] The word comes from a Japanese phrase meaning, roughly, "empty orchestra".

ACKNOWLEDGEMENTS

Thanks to the many authors who have gathered together and sorted out in other publications much of the minutiae of the MIDI interface and language, from which has come much of my own understanding of the topic.

Thanks to my bride, Carla Red Fox, for her insightful copy editing of this rather tedious manuscript. Her Cherokee name might almost better be Carla Red Pencil, as she has counted many coup with that stick.

#

APPENDIX B
MIDI MESSAGES IN DETAIL

INTRODUCTION

The MIDI language is defined in terms of MIDI messages, which are organized in a taxonomy of classes.

In this appendix, we:

- Summarize the taxonomy of MIDI messages
- Review the basic of MIDI message structure and the notation we will use to describe message details
- List almost all MIDI messages with details of their format, coding, and syntax.

Keep in mind that not all MIDI-oriented sound modules (*e.g.*, synthesizers) will have all the capabilities supported by the rich repertoire of MIDI messages, and accordingly may not respond to all message types (usually, by a long shot).

MESSAGE TAXONOMY

The range of MIDI messages is divided into five classes.

Channel voice messages

These all carry a channel tag. They include these types and type families (among others):

- Note messages. These start and stop the sounding of notes.
- Program change message. This advises the synthesizer (or other sound module) which "patch" ("instrument sound) to use to render the notes controlled by note messages on the channel over which this message arrives.
- Control change messages. This is actually a familiy of many messages, each of which controls some aspect of the behavior of the synthesizer, with respect to notes sounded by note messages on the channel over which the control change message arrives. An illustrative example is the control of volume for all such notes, or the invocation of the piano "sustain pedal".

There are other types in this class.

Channel mode messages

This class has a misleading name. These are "global" messages that affect the behavior of a synthesizer or other sound module with respect to notes arriving over any channel.

Nevertheless, these messages have a channel tag. This is to cater for the possibility of more than one sound module "riding" the same message stream. Each of those modules has a certain channel (set by the user, called the "basic channel") on (only) which it will listen for "channel mode messages". We would set this to a different channel on the different sound modules. Then, the channel mode messages intended for each module are tagged for the module's "basic channel".

This class includes the following subclasses (among others):

- "Stop sounding" messages. These tell the sound modules to stop sounding, and are usually used in case some confusion leaves the sound module sounding some notes when we don't want it to. One calls for a graceful shutdown, in which notes stop in the usual way, perhaps with a decay of sound, or a "legato hangover", as applicable at the time. The other makes the module go completely quiet immediately.

- Reset all controllers message. This sets to "normal" the collection of behavioral parameters that can be varied by Control Change messages.

- Channel configuration mode messages. These are used to establish four different paradigms of how the sound module responds to messages arriving over different channels.

System real-time messages and system real time messages

These messages primarily include those used to actually control sequencers (units that send MIDI message streams to cause a performance), and to provide a clock to keep multiple sequencers in synchronism.

They carry no channel tag.

System exclusive messages

Initially, this class provided for "proprietary" messages used by individual manufacturers of synthesizers to control specialized features. Today, there is some provision for standardization of "advanced" messages sent in this way.

They carry no channel tag.

MIDI MESSAGE FORMAT

Structure

With the exception of the "System Exclusive" (SysEx) message type, which has an open-ended variable length, all MIDI messages comprise one, two, or three consecutive bytes.

At the electrical interface, these bytes are sent as serial asynchronous characters. The eight data bits in these characters are sent least-significant bit first (as is done for the serial transmission of ASCII characters).

There is no issue here of "big-endian" vs. "little-endian" ("Intel vs. Mac") sequence of multiple bytes. The bytes/characters are sent in the sequence described.

Each message comprises one *status byte*, followed by zero, one, or two *data bytes*. The most significant data bit of a status byte is always "1", while for a data byte, it is "0".

In the status byte of a message with a channel tag (see below), the three bits below the most-significant bit (that is, below in order of significance) indicate the "message type". In messages without a channel tag, the remaining four bits of the status byte (four least significant) complete the identification of the message type.

MIDI channels

Except for a few message types, every MIDI message carries what we may call a *channel tag*, coded in the least-significant four bits of the status byte. This tagging scheme in effect creates 16 distinct "channels" over which messages may be considered to travel. This allows one message stream to carry, on a multiplexed basis, messages intended for separate physical devices (all "riding" the same stream, perhaps through a daisy-chain wiring arrangement) or intended for separate "parts" of a multitimbral synthesizer. Although the channel code ranges from 0-15 (0000 through 1111), the *channel number* is considered to range from 1-16.

Describing the coding of the status byte

Although a three-bit subfield of the status byte gives (in most cases) the message type, we do not usually quote that three bit value in any form. More likely, we will give the hexadecimal value of the whole byte (including the most significant bit, always one), perhaps with a variable symbol to represent the lower-order "nibble" Thus, for a Note On message (the first type we will encounter in our encyclopedia), we

may say that the data byte is 9*m*, where *m* represents the MIDI channel tag value ("MIDI channel", or just "channel", for short).

In fact the actual message type code for the Note On message is 1_{hex} (001_{bin}). But in what we usually see, it looks as if it is 9_{hex} (1001_{bin}) because the "status bit flag" is included in that reckoning. We rarely see mention of the "bare" message type code.

Numerical values

We will give actual values of the bytes of each message (or of four bit "nybbles" of a byte) in hexadecimal form (usually without any indicator of the radix). Where meaningful, we will give the corresponding decimal expression of the value afterwards in square brackets.

CHANNEL VOICE MESSAGES

Note On message

This message tells the sound module to begin sounding a note.

Three bytes: 9*m nn vv*

where *m* is the MIDI channel code, *nn* is the note number, and *vv* is the note velocity. (It is best to think of *nn* and *vv* as seven-bit numbers, since the most-significant bit of their bytes is always 0, the marker of a data byte.)

The note number tells the pitch of the note to be sounded. The value 3C [60] corresponds to middle C. The increment is a half step (semitone), with greater values representing higher pitch. The range is 0-7F [0-127] Any value in the range is valid. (Of course, the note specified may be outside the range of the "instrument" being addressed, and thus may not be rendered.)

The note velocity indicates the loudness of the note, but may also have implications on its timbre. The term is drawn from the concept of the velocity with which a piano key is struck.

Note Off message

This message tells the sound module to cease sounding a note.

Three bytes: 8*m nn vv*

where *m* is the MIDI channel code, *nn* is the note number, and *vv* is the note velocity.

The note number tells the pitch of the note to be ceased.

The note velocity potentially indicates the rapidity with which the note is to be ceased ("release velocity"), but is meaningless in many cases. It is the convention to use the value 40 [64] if this parameter is not meaningful.

Alternate convention for the Note Off function

If a Note On message is sent with its velocity parameter, *vv*, having the value zero, that is to be taken as an instruction to stop sounding the note indicated by the note number, *nn*. Clearly the subtlety of "release velocity" is foregone when using this convention. This convention is useful in connection with a "shorthand" syntax for the representation of a string of messages of the same type, used for transmission efficiency (the "running status" syntax).

Key Pressure ("aftertouch") message

This message indicates that the pressure on the key causing a note to sound has changed after the initial "strike" (for systems that can vary the loudness of the note while it is sounding).

Three bytes: A*m nn xx*

where *m* is the MIDI channel code, *nn* is the note number of the key, and *xx* is the new "pressure".

Keyboard Pressure message

This message indicates that the pressure on the entire keyboard (with respect to notes on the referenced channel) has changed.

Two bytes: D*m xx*

where *m* is the MIDI channel code and *xx* is the new "pressure".

Pitch Bend message

This indicates that the pitch of all notes (initiated on the reference channel) now being sounded should change (as done with a "pitch bend wheel" on a keyboard instrument, a "pitch bend lever" on a guitar, etc.) The resolution is 14 bits.

Three bytes: A*m ll hh*

where *m* is the MIDI channel code, *ll* carries the 7 least-significant bits of the 14-bit value, and *hh* carries the 7 most-significant bits. "Neutral" is given by *ll*,*hh* = 00,40 (essentially the center of the range of a 14-bit number).

The "sensitivity" of this control (that is, how many units of change makes a pitch change of one-half step) may be set with a message of the "registered parameter number" family.

Program Change (PC) message

This message tells the sound module to adopt a certain "patch" (timbre definition) for all notes to be subsequently rendered in response to Note On messages received over the same channel.

Two bytes: C*m pp*

where *m* is the MIDI channel and *pp* is the program number code.

The range of *pp* is 0-7F [0-127). However, in most (but not all) cases, the program number itself is considered (for humans) to have the range 1-128.

Note that the basic message syntax does not define what patch is to be put in effect for a certain program number, that being left up to the implementer. But (as mentioned in the body of the article), there are standardized lists of patches with explicit associations to program numbers.

Control Change (CC) messages

This is actually a family of messages, each of them setting some property of the way in which the synthesizer is to render the notes received over the same channel as the CC message—essentially, the command of a "controller" (such as the sustain pedal of a piano).

Two bytes: B*m cc xx*

where *m* is the MIDI channel, *cc* is the controller number, and *xx* is the controller value.

For some controllers ("switched" controllers), there are actually only two values that can be sent, "off" and "on". For those, any value of *xx* from 0-3F is interpreted as "off", and any value of *xx* from 40-127 is interpreted as "on". It is however considered good etiquette to actually send *xx* with only the values "0" or "127". In any case, it is only the most-significant data bit (of the seven data bits of the second data byte) that conveys the "off" *vs*. "on" indication.

For other controllers ("continuous" controllers), the *xx* value sent can range from 0-127. If the value implies variation above and below some "neutral" point, that point is usually represented by the *xx* value 40.

In many cases, the actual value to be set may (in some implementations) be given in 14-bit precision. To do so, two different CC messages are used. In one, the 7 data bits of data byte 2 are the seven lowest-order bits of the complete 14 bit value, while in the other, the 7 data bits are the highest-order bits of the complete 14 bit

value. The two messages are distinguished by the terms "LSB" and MSB" in their names (for *most significant byte* and *least significant byte*, respectively, although those are misnomers).

In general, if it is not needed to send the value to 14-bit precision, only the "MSB" message is used. If, after initially sending both MSB and LSB components, a change is needed that would only affect the lower-order seven bits, just a new LSB message may be sent. (This does not apply to the "bank select MSB" and "bank select LSB" messages. See detailed discussion in the body of the article.)

The various members of the control change family are listed below, identified by the *cc* code in hexadecimal.[20] Where there are MSB-LSB pairs (for 14-bit resolution capability), the two messages are shown together even though the *cc* values are not adjacent.

Continuous controllers with 14-bit resolution capability

00/20 Bank select MSB/LSB (see discussion in body of paper)

01/21 Modulation wheel MSB/LSB

02/22 Breath controller MSB/LSB

04/24 Foot controller MSB/LSB

05/25 Portamento time MSB/LSB

07/27 Channel volume MSB/LSB

08/28 Balance MSB/LSB (symmetrical—neutral is 40/00—for symmetry, the range is 01/00 to 7F/00)

0A/2A Pan position MSB/LSB (symmetrical—neutral is 40/00—for symmetry, the range is 01/00 to 7F/00)

0B/2B Expression MSB/LSB

0C/2C Effect control 1 MSB/LSB

0D/2D Effect control 2 MSB/LSB

10/30 General purpose controller 1 MSB/LSB

11/31 General purpose controller 2 MSB/LSB

12/32 General purpose controller 3 MSB/LSB

13/33 General purpose controller 4 MSB/LSB

Switched controllers

40 Sustain (damper) pedal

---

[20] Note that when we concisely identify a control change message in the form "CC32", the numerical portion is the *cc* value expressed in **decimal** form.

41 Portamento (off/on)

42 Sostenuto (off/on)

43 Soft pedal

44 Legato footswitch

45 Hold 2

46 Sound controller 1 (Default: Sound Variation)

47 Sound controller 2 (Default: Timbre/Harmonic Intensity)

48 Sound controller 3 (Default: Release time)

49 Sound controller 4 (Default: Attack time)

4A Sound controller 5 (Default: Brightness)

4B Sound controller 6 (No default defined)

4C Sound controller 7 (No default defined)

4D Sound controller 8 (No default defined)4E Sound controller 9 (No default defined)

4D Sound controller 8 (No default defined)

Continuous controllers (7-bit resolution)

50-53 General-purpose controllers 5-8

54 Portamento control (*xx* is the note number of the note from which to start the "slide". For the next Note On message, the note slides from this value to the note value in the Note On message itself. The time over which the slide occurs is set by the appropriate control change message or message pair, with *cc*=05/25)

5B Effect 1 depth (Default: External Effects depth)

5C Effect 2 depth (Default: Tremolo depth)

5D Effect 3 depth (Default: Chorus depth)

5E Effect 4 depth (Default: Celeste depth—detune)

5F Effect 5 depth (Default: Phaser depth)

The remaining controllers are highly specialized; the specifics of their operation and use are beyond the scope of this article.

06/26 Data entry MSB/LSB

60 Data increment

61 Data decrement

62/63 Non-registered parameter number (NRPN) MSB/LSB

64/65 Non-registered parameter number (RPN) MSB/LSB

CHANNEL MODE MESSAGES

All Notes Off message

This message terminates all notes being sounded by the synthesizer (but see discussion of subtleties below).

Three bytes: B*m* 7B 00

where *m* is the MIDI channel code. It must be the *basic channel* established for the particular synthesizer.

The notes are terminated as if Note Off messages had been received for all notes now sounding. That means that their normal ending decay envelope is followed, any reverberation in effect will play out, and if the "Hold" function is on, the notes will still sound.

All Sound Off message

This message **immediately** terminates all notes being sounded by the synthesizer.

Three bytes: B*m* 78 00

where *m* is the MIDI channel code. It must be the *basic channel* established for the particular synthesizer.

Reset All Controllers message

This message resets all controller settings to their normal or default positions

Three bytes: B*m* 79 00

where *m* is the MIDI channel code. It must be the *basic channel* established for the particular synthesizer.

Local Control message

This disables or enables the path from the keyboard of a keyboard instrument to its internal synthesizer.

Three bytes: B*m* 7A *xx*

where *m* is the MIDI channel code. It must be the *basic channel* established for the particular synthesizer.

The value *xx* controls the off/on status: a value in the range 00-3F indicates off and a value in the range 4F-FF indicates on (it is preferable to send either 00 or FF).

Channel mode configuration messages[21]

These four messages control how the synthesizer responds to notes on various channels. The details are beyond the scope of this article.

The four messages are:

a. Omni Mode Off—three bytes: B*m* 7C 00
b. Omni Mode On—three bytes: B*m* 7D 00

(The two above are alternatives—one or the other must be in effect.)

c. Mono Mode—three bytes: B*m* 7E 00
d. Poly Mode—three bytes: B*m* 7F *xx*

(The two above are alternatives—one or the other must be in effect.)

Any of the four combinations of either a or b in effect and either c or d in effect constitutes a *channel mode*.

The most common arrangement (and most of the discussions in this article are predicated on it) is to have **Omni Mode On** and **Poly Mode** in effect (channel mode 1).

SYSTEM EXCLUSIVE MESSAGE

This is the only variable length MIDI message. It is primarily used in a proprietary context.

The message structure is:

Start of System Exclusive message—One byte: F0

Data bytes (any number, all with most significant bit = 0)

End of System Exclusive message—One byte: F7

If the System Exclusive message is to be followed promptly by any other message (other than a System Real Time message), the End of System Exclusive message indicator may be omitted.

The first data byte, or the first three, carry a manufacturers' identification code (three-byte codes have 00 as the first byte).

SYSTEM REAL TIME AND COMMON MESSAGES

These are highly specialized. Only a few will be described here.

---

[21] These are often called just "channel mode messages", but they are only part of the class called "channel mode messages", so the notation is rather ambiguous.

Their status bytes have values in the range F1-FF. They have zero, one, or two data bytes (determined by the message type).

The ones we list are only those involved in the control of a subordinate MIDI sequencer.

Real time messages

MIDI Clock message

This message provides a steady stream of "clocks" to control the pacing of a "subordinate" sequencer. It is sent 24 times per quarter note (once per 16th note).

One byte: F8

MIDI Start Message

This message makes a subordinate sequencer commence play, from the beginning of the current "song", upon the arrival of the next MIDI Clock message.

One byte: FA

MIDI Stop Message

This message makes a subordinate sequencer cease play. The subordinate sequencer sets the value of its internal song position pointer to the next "clock position" (units of one 16th note) after the cessation of play, in anticipation of the possibility of resuming play at that point by command of a MIDI Continue message.

One byte: FC

MIDI Continue Message

This message makes a subordinate sequencer commence play, from the beat indicated by the current value of the subordinate sequencer's song position pointer, upon the arrival of the next MIDI Clock message.

One byte: FB

System Common message

MIDI Song Pointer Position message

This message forces the song position pointer in the subordinate sequencer to a certain value, in anticipation of play being commenced at that point by command of a MIDI Continue message.

Three bytes: F2 *ll hh*

where *ll* carries the 7 least-significant bits of the 14-bit parameter value, and *hh* carries the 7 most-significant bits.

One unit of the parameter of this message corresponds to 6 MIDI timing clocks, or 1/4 of a quarter note, or one 16th note. A value of zero implies the very beginning of the song (the beginning of the first beat of the first measure).

#