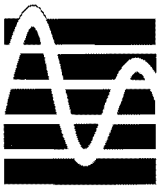
Volume 7 • THE COMPUTER MUSIC AND DIGITAL AUDIO SERIES



MIDI

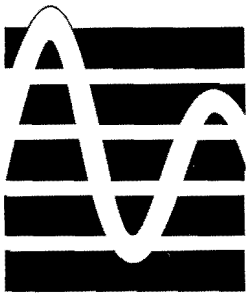
A COMPREHENSIVE INTRODUCTION

2nd Edition

Joseph Rothstein

A-R Editions, Inc.

Madison, Wisconsin



ONE

Overview

MIDI is an acronym for "Musical Instrument Digital Interface." Despite its impact on the world of music, MIDI is not a musical language, nor does it directly describe musical sounds. Rather, it is a data communications protocol, an agreement among manufacturers of music equipment, computers, and software that describes a means for music systems and related equipment to exchange information and control signals.

The difference between audio information and MIDI data is similar to the difference between a tape recording of a pianist performing a piano sonata and the sheet music for that same sonata. The recording captures the musical information itself, storing the actual sounds that came from the piano.

Data, on the other hand, symbolically represent information for the purposes of storing or transmitting it in a more compact or more easily transmitted form. Thus, each notational symbol in printed piano music stores not the sound of the piano music but the instructions necessary for a human performer to re-create that music. The performer converts the symbol back into the information it represents by striking the key indicated by the note symbol, using the indicated amount of pressure, and holding the key for the indicated length of time. Notational symbols in printed music are thus forms of data.

MIDI represents yet another level of abstraction from piano music. Whereas printed piano music symbolically describes musical sounds, MIDI symbolically describes the electronic steps required to generate the sounds. That is, we can think of MIDI as specifying exactly which electronic circuits should turn on and exactly how

long they should remain on. Printed music describes a musical result that a human performer must interpret and re-create.

The MIDI protocol thus enables us to manage the complex information representing the performance of a musical work on electronic instruments. MIDI represents the information needed to re-create a performance as many individual pieces of data. The MIDI protocol specifies the meaning of each data value and provides a means to store, manipulate, transmit, and re-create the information using symbolic data.

MIDI is important to anyone involved in music making using digital electronics and increasingly so to anyone involved in any art form using digital electronics. The significance of MIDI is that it permits a wide variety of equipment from many different manufacturers to work together in a single system. Each piece of equipment is adhering to the same rules of information management specified by MIDI so that they have a common system of communication.

■ MIDI APPLICATIONS

One of the simplest and most widely used applications of MIDI is to connect two or more synthesizers together so that playing one produces the same note on each. That way, a player at a single MIDI keyboard or other controller can produce sounds that have a rich, layered, complex timbre rather than a single, easily identifiable sound. MIDI devices other than just synthesizers can become part of that single system under the control of a single player. There are now MIDI-controlled digital effects processors, mixing consoles, and lighting control panels. As a result, a player at a master MIDI controller can run a complex multimedia setup of lights, sound, and effects by sending MIDI messages from the master controller to the correct device.

More powerful applications of MIDI technology involve a computer, making it an integral part of the MIDI system. The computer can be used to record MIDI messages, much as a tape recorder records audio signals. Because MIDI messages are digital data, MIDI recordings can use the full power of digital computers. With appropriate software, the computer can simulate a multitrack tape recorder using not just 4, 16, or 48 tracks but hundreds of tracks, and in some cases, unlimited numbers of tracks.

Rather than simple recording, playback, and cut-and-splice editing, a MIDI recorder includes powerful facilities that provide the user a tremendous range of creative controls. Certainly, MIDI recordings can be edited—to a precision of hundredths of a beat. In addition, the pitch, rhythm, tempo, or repetition of a single note, a longer passage, or an entire piece can be adjusted quickly and accurately. Each track can hold musical information intended for a particular synthesizer or control information for any MIDI device. The same recording that plays a synthesizer track can control an effects processor or a mixing console, setting reverb, equalization, or mix levels with a precision and accuracy that would be impossible otherwise.

Some MIDI programs are intended specifically for composing. These "intelligent instruments" let composers arrange musical ideas, concepts, and structures, with the computer making some (or many) of the specific decisions about notes, chords, and rhythms. The composer can hear the results as they occur and make changes as the music plays. As programmers apply artificial intelligence concepts to music composition software, the programs are likely to gain a wider range of capacities.

Other computer programs translate MIDI data into printer control data so that the computer becomes a powerful composer's and arranger's assistant. The capabilities of MIDI scoring programs vary widely. However, most such programs will let the composer enter music from a MIDI controller or the computer's typewriter-style keyboard. The composer can also enter tempo, meter, and expression markings and print out the completed score on a graphics printer.

With so many synthesizers and so much complexity, it is only natural to use computers to cut down the volume of data that synthesizer operators must remember. One approach uses patch librarian software, which stores synthesizer settings with appropriate notes and comments, then transmits selected patch data to the correct synthesizer at the correct time. In this way, the synthesizer operator need not be concerned with setting parameters from scratch each time the patch is used but need only select the desired patch from the library.

And, of course, the computer is an ideal tool for education. It is patient, precise, and always available. Music education programs are available for ear training, sight-reading, keyboard skills development, and music theory and are designed for a wide range of skill levels.

THE MIDI SPECIFICATION

The MIDI protocol specifies conditions in two areas, and manufacturers must meet these conditions to call their product a MIDI device. The areas are the hardware interface, or the physical connection between two separate pieces of equipment, and the data format, or the arrangement and order of data messages that one device transmits to another. The hardware portion of the MIDI specification describes the two parts of the physical connection between different pieces of equipment. The two parts are, first, a MIDI port that converts MIDI data from its digital form to the series of electrical voltages that represent the MIDI data numbers (or vice versa) and, second, a MIDI cable that transmits the voltage signals to the next device's MIDI port, which converts the voltages back to digital data.

MIDI Ports

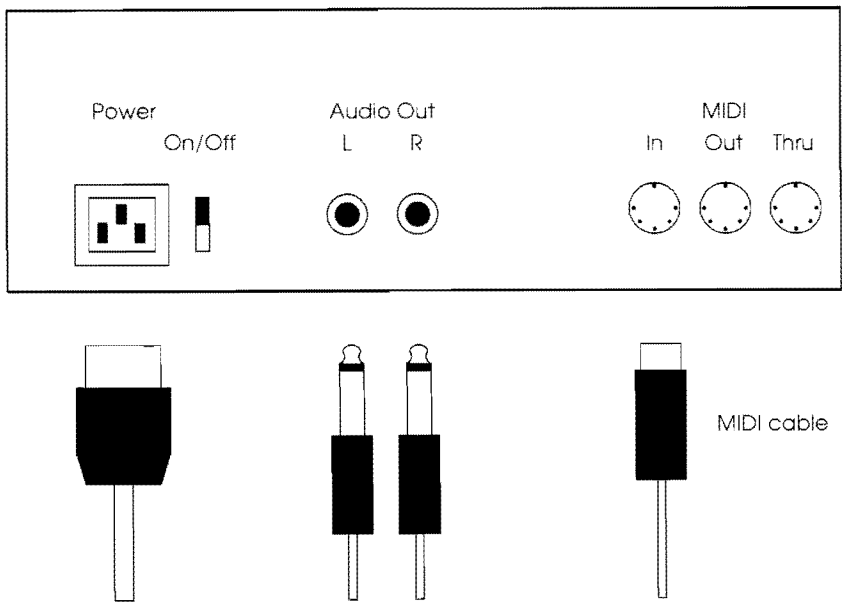
A port, or interface, is a place where two otherwise incompatible systems perform the necessary conversions that enable them to pass data from one to the other. Computers use groups of electrical voltages to represent large numbers. The MIDI cable carries voltages representing only the numbers 0 and 1. Conversion from one form to the other takes place at the MIDI port. Although "MIDI interface" is redundant, it is more commonly used than "MIDI port."

The MIDI specification describes three types of ports, each represented by a female jack designed to receive the five-pin MIDI cable connector, as shown in figure 1.1. At the MIDI Out jack, data are converted from digital to voltage format. The voltages travel along a MIDI cable to another MIDI device. The receiving device must include a MIDI In jack to convert incoming voltages back to their digital format. It is important when connecting MIDI devices to be sure which is sending signals and which is receiving them and to connect the cables accordingly.

A simple example involves using a "master" keyboard synthesizer to control a "slave" keyboard or rack-mount synthesizer so that playing the master keyboard will cause the two to sound in unison. The MIDI cable should plug into the master's MIDI Out jack and the slave's MIDI In. As obvious as this may seem, incorrect cabling is one of the most common causes of MIDI system problems. Check for proper cabling first when such problems occur.

Most MIDI devices have both MIDI In and MIDI Out jacks, but there is a third port defined in the MIDI specification called MIDI

Figure 1.1 Rear-panel connections on a typical MIDI device, showing the location of MIDI In, Out, and Thru ports.



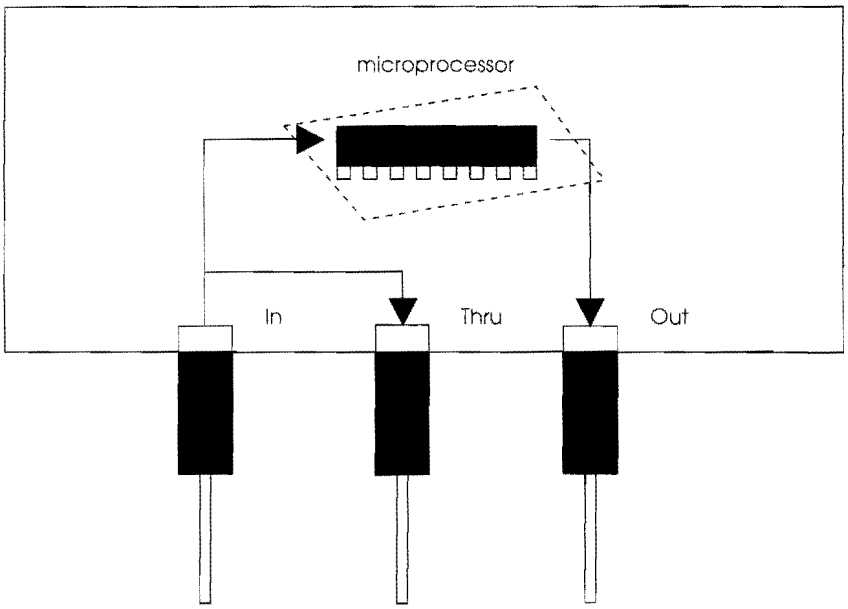
Thru. It makes a copy of any incoming signals the device receives through its MIDI In port and immediately retransmits them, unchanged, out the MIDI Thru jack (figure 1.2). In a sense, MIDI Thru serves as a Y-connector, which breaks a signal in two. Chapter 7, "Getting It All to Work Together," covers the advantages and pitfalls associated with using MIDI Thru.

MIDI Cables

A MIDI cable consists of a length of wire with identical connectors at each end, as shown in figure 1.3. The cable transmits neither audio signals nor electrical power. Rather, the changes in voltage transmitted through the MIDI cable represent binary data: the yes/no, on/off signals that, strung together, represent specific numbers whose purpose and meaning are defined in the MIDI protocol.

MIDI cables are generally no longer than 50 feet and usually are much less than that. Voltages become weaker the farther they

Figure 1.2 MIDI In, Out, and Thru ports, indicating the direction of data flow.



travel, so to avoid potential problems it is a good idea always to use the shortest possible MIDI cable.

Most MIDI cables use five-pin DIN connectors, one at each end of the cable. European electronic equipment manufacturers have long used DIN connectors; they are inexpensive and readily available. The current MIDI specification uses only three of the five pins, and most manufacturers have left the remaining pins unused in case some future addition to the MIDI specification defines uses for them. Still, a few manufacturers have used the two pins for their own purposes. The documentation of any piece of MIDI equipment should indicate whether the equipment uses more than three pins. If so, try to find an alternative that uses only the three pins specified by MIDI.

One problem with DIN connectors is that they are not particularly durable and may break after repeated disconnecting and reconnecting. Many manufacturers of professional audio equipment use a tougher but more expensive three-conductor XLR connector. The MIDI specification permits manufacturers to use XLR connectors, provided that they also furnish adapters for connecting them to DIN connectors.

Figure 1.3 MIDI cable—note the identical connectors at each end (courtesy Hosa Technology, Inc.).

MIDI cables themselves can also be a source of problems if they should pick up stray electrical signals given off by electronic equipment. Better-quality MIDI cables wrap the wiring inside a thin aluminum sheath before wrapping the entire cable in an outer covering of plastic or rubber. The aluminum shielding cuts down on the cable's receptivity to such unwanted signals, so an important rule for any MIDI setup is to use only shielded cables. This goes for audio cables as well as MIDI cables, as cheap cables sabotage too many expensive setups. It is impossible to tell by its outside appearance whether a cable is shielded. If you make it a rule to buy only shielded cables, you will have one less thing to worry about later. Remember, if the price is cheap, the cable probably is, too! (Because the humble cable is the source of so many potential problems, there is a lot more on cabling in chapter 7.)

MIDI MESSAGES

Each of the different number sequences in the MIDI data format specification is called a MIDI message. Each message describes a particular event, for example, the start of a musical note, the change in a switch setting, the motion of a foot pedal, or the selection of a sound "patch."

Because MIDI specifies a digital (i.e., numeric) data format, every MIDI device must contain at least one microprocessor, a "computer on a chip" that can interpret and act on MIDI messages. The microprocessors in MIDI synthesizers and other MIDI-controlled equipment make sense of the vast amounts of numbers that MIDI uses to describe the processes required to create (or re-create) a piece of music.

Musicians generally think of music as a series of notes, each one having a particular pitch, amplitude, and timbre. Musicians thus consider a note the most basic of musical messages. Likewise, the most basic MIDI message is the Note On message, though a Note On message differs from our usual understanding of a musical note. Both contain pitch and loudness data, but MIDI treats elements of expressiveness such as vibrato and pitch bend as separate MIDI messages rather than as part and parcel of each note that they affect.

A performer playing a MIDI instrument can generate many types of MIDI messages. The messages might include on/off switch settings or continuously varying controller information such as modulation, which changes the timbre, or tone color, of the note.

A device or computer program called a sequencer can store and play back sequences of messages. The messages may specify notes, so that different MIDI devices know which notes to play and how loud to play them. Other messages may request that a particular MIDI synthesizer change from one instrumental sound to another. The sequencer stores the messages, maintains the time relationships among the various messages, and transmits the stored messages at the appropriate times. MIDI devices connected to the sequencer re-create the performance specified by the stored messages.

The developers of the MIDI specification could not foresee all the ways that musical instruments might develop and change. Thus they left a way for individual manufacturers to control specific aspects of their equipment that the MIDI specification left undefined. System Exclusive messages apply only to particular equipment from a par-

ticular manufacturer. The specifics of the System Exclusive messages are defined by each manufacturer and are ignored by equipment from other manufacturers.

Understanding MIDI messages is crucial to making full use of MIDI. Chapter 8, "Understanding MIDI Messages," describes each type in detail.

A BRIEF HISTORY OF MIDI

Using electronic technology to produce music has been around as long as electronic technology itself. Though such electronic musical instruments appeared as early as the beginning of the twentieth century, the first popular electronic synthesizers were probably the Theremin and the Ondes Martenot. Their unique sound qualities were especially appropriate in dramatic settings, and numerous movie scores of the 1920s and 1930s featured their synthesized sounds.

As vacuum-tube and then transistor technology evolved, musicians were quick to put the technology to use. The analog synthesizers that resulted represented sound as a constantly changing electrical voltage whose charge was analogous to the sound pressure the voltage represented. Such synthesizers of the 1940s, 1950s, and early 1960s were generally large, expensive to build and maintain, and difficult to operate. Apart from an occasional movie score or public exhibition, analog electronic synthesizers were confined mostly to academic research institutions, music conservatories, and the studios of avant-garde composers.

The first mass-market analog electronic synthesizer was designed by Robert Moog in the 1960s. A 1968 recording of the music of Johann Sebastian Bach performed on a Moog synthesizer by Wendy Carlos set the recording industry and listening public on its ear. *Switched-On Bach* became one of the best-selling classical music albums of all time, and it continues to sell well, a generation later.

As millions of people became enchanted by the unique sounds used in *Switched-On Bach,* Moog became a household name, even if it was consistently mispronounced (it rhymes with "vogue"). Many of *Switched-On Bach*'s young listeners formed rock bands, and many of those bands used Moog synthesizers along with a flood of analog synthesizers that followed from ARP, Sequential Circuits, and other manufacturers.

However, problems still beset the newer analog synthesizers. Though smaller in size than their earlier counterparts, they were still difficult to set up and operate. "Programming" such synthesizers involved using patch cables to make temporary electrical connections among the synthesizer's various sound-generating oscillators, filters, and other components.

Most synthesizers of that era were monophonic; that is, they could play only one note at a time. It was a common sight at concerts to see a keyboard player surrounded by a wall of synthesizers, each with its own tangle of patch cables and a bewildering array of buttons, switches, and sliders. To make matters worse, the tuning of analog synthesizers was prone to drift (particularly as their components heated up). Every few minutes or so, the entire system had to be retuned, one synthesizer at a time. Furthermore, multiple synthesizers meant multiple keyboards, multiple oscillators, multiple patch-cord setups, and multiple headaches.

Computer music synthesizers, in which sounds are represented by numbers within a computer's memory, had also been around for almost two decades by the early 1970s. However, the high price of computer components had kept them out of reach of most musicians.

When low-cost microprocessors and integrated circuits began to appear, musical instrument manufacturers eagerly adopted them. Digital circuits are inherently more stable, more reliable, and cheaper (when produced in large quantities) than their analog counterparts. Replacing analog oscillators and other voltage-based components with digital microchips resulted in more reliable, cheaper, and "smarter" synthesizers.

By the end of the 1970s many manufacturers had begun to offer digital synthesizers. The tangles of patch cords—the hallmark of analog synthesizers like the early Moog designs—began to disappear. They were replaced by control panels and software that allowed the user to change voices at the touch of a key. Furthermore, once they leave the factory, most digital synthesizers never need retuning.

However, the problem of duplicate components (especially keyboards) remained. Keyboards are mechanical and therefore more expensive than similarly complex electrical devices. Because each manufacturer used a different design scheme, each synthesizer had its own keyboard and control panel. Using a wide range of synthesizer sounds still meant a wall of equipment and the prospect of dashing from one keyboard to the next. Musicians were still limited to playing only one or two synthesizers at a time

or having multiple players for the multiple keyboards, neither one an appealing choice.

In 1981, three synthesizer company employees decided to do something about it. Dave Smith of Sequential Circuits, I. Kakehashi of Roland Corporation, and Tom Oberheim of Oberheim Electronics met during that June's show held by the National Association of Music Merchants (NAMM). They discussed ways that synthesizer manufacturers might standardize their control signals so that different companies' synthesizers might work together.

Because the new synthesizers were really digital computers disguised as musical instruments, these employees considered the ways that computer manufacturers had addressed the same challenges of intermachine communication. One of the most successful approaches is the Local Area Network (LAN), a hardware and software system that permits computers from different manufacturers to share data and equipment such as printers. Local Area Networks had many of the characteristics the synthesizer group wanted. They can be relatively simple and inexpensive to set up and run, they can be nonhierarchical (i.e., they can handle all the computers in the network on an equal basis), and they clearly define the hardware requirements for physically connecting the computers and the data format of the messages that pass among the computers. Dave Smith wrote up the proposal, called the "Universal Synthesizer Interface," and presented it at the November 1981 meeting of the Audio Engineering Society.

During the next NAMM show in January 1982, such major Japanese synthesizer manufacturers as Yamaha, Korg, and Kawai joined the original companies. The new, larger group refined and expanded the proposed standard. By the June 1982 NAMM show, the basics were in place for what was to become MIDI: the Musical Instrument Digital Interface. Companies began developing synthesizers according to the early MIDI specifications, and developers used those early efforts to test and further refine the MIDI specification on the basis of real-world experience.

The full *MIDI 1.0 Detailed Specification* was first released to the general public in August 1983. The International MIDI Users Group (IMUG) was formed and given the responsibility for distributing the MIDI specification to musicians and music manufacturers. This group soon evolved into the present-day International MIDI Association (IMA), whose primary purpose is to be a clearinghouse for information about MIDI. (For more about the IMA, see chapter 11, "Getting Help.")

The manufacturers, concerned that they would lose control over MIDI, formed the MIDI Manufacturers Association (MMA) to safeguard the MIDI specification and implement changes in an orderly fashion. Together with the Japan MIDI Standards Committee (JMSC), the MMA controls the MIDI specification. Before any changes can be adopted as part of the evolving MIDI specification, both the MMA and the JMSC must review and agree on the proposals.

LIMITATIONS OF MIDI

MIDI is not without its shortcomings, however. One is that the MIDI specification does not describe what MIDI devices must do but only how to communicate something if the manufacturer decides to do it. Few MIDI devices will incorporate all the potential power of MIDI, so it is up to the person who buys and uses a MIDI device to find out what it can and cannot do. Capabilities range from simple one-voice Note On/Note Off generators to complicated systems with touch sensitivity, keyboard splits, and modulation control.

It should be no surprise that, within limits, the price of an instrument offers as reliable an initial guideline to its capabilities as any. Even though prices of MIDI equipment seem to drop as technological capabilities increase, there will probably always be disparities between expensive and inexpensive instruments and even between similarly priced equipment. "Caveat emptor," as the Romans used to say: Buyer beware.

Nor does MIDI specify particular sounds that an instrument must make or the quality of the sounds that it does make. The range and quality of sounds from MIDI synthesizers vary from numerous and wonderful to few and awful, and MIDI itself will not help you determine which synthesizer has which.

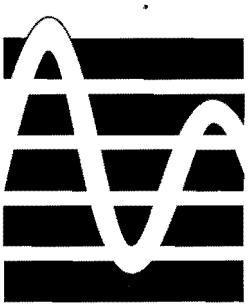
Those are artistic limitations of MIDI, but there are technical limitations as well. MIDI sends data serially (i.e., one small piece of each message at a time), and the rate at which MIDI messages travel is relatively slow (compared to some other data transmission protocols). If many devices in a single MIDI setup are each generating many messages, all at the same time, the tremendous number of simultaneous MIDI messages can overwhelm MIDI's ability to handle them, a condition called *MIDI clog*.

In complicated setups, there can be an audible delay, called *MIDI lag*, between the time a message is initiated and the time you hear the result of the message. Our ears are sensitive instruments, and we can sense (even if not consciously hear) delays in the range of thousandths of a second.

Finally, MIDI provides for only sixteen channels of data. As more devices use MIDI for their control signals, the limit of sixteen channels has become inadequate for many complex setups. With one channel devoted to your mixer, another to your digital effects processor, and another to your equalizer, there are not nearly enough for that forty-voice orchestral spectacular you have been dreaming of—and that does not even count the channel you need to control the lighting panel!

Fortunately, the MIDI specification has room to grow and develop, though its development will be tightly controlled to prevent a tangle of incompatible "innovations." Already, multiport interfaces are available; with the right software, they can handle more than sixteen channels without defying the MIDI specification. (For more on multiport interfaces, see chapter 4, "MIDI Hardware—How to Choose It, How to Use It.") Products that point toward solutions of MIDI's transmission-speed limitations have also begun to appear.

Despite MIDI's limitations and shortcomings, it has changed the way many musicians work and indeed the way they think about music making. It is safe to say that no new technology has so changed the musical landscape since the introduction of the piano, and it is likely that MIDI's effects will be equally widespread and long-standing.



EIGHT

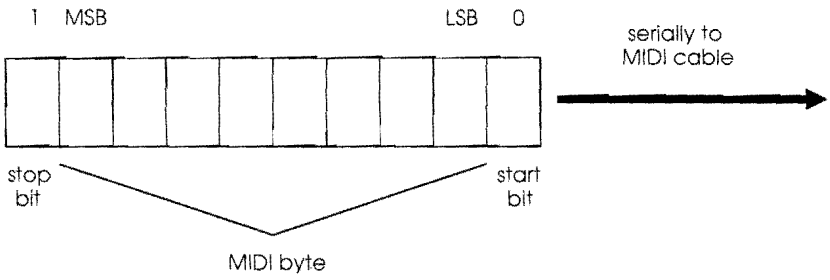
Understanding MIDI Messages

MIDI messages are the means by which one MIDI device communicates information to another MIDI device. Each message specifies a musical event—to understand why a particular device acts in a particular manner, you must understand what messages it is receiving and transmitting. Most sequencers, and some other devices, can display MIDI messages (often in a variety of formats), allowing sequencer users to examine and alter MIDI messages one by one.

MIDI messages consist of one or more bytes of digital data. A MIDI cable carries voltages representing digital 1s and 0s: the binary digits, or bits, that are the building blocks of computer data. Audio inputs and outputs receive and transmit continuously varying voltages representing analog sound data. Consequently, the output from a MIDI port should never be connected to an audio input, nor should an audio output be connected to a MIDI In port. Damage to both pieces of equipment could result.

Each MIDI message byte begins with a start bit (logical 0), followed by eight message bits sent with the least significant bit first. Each message ends with a stop bit (logical 1), for a total of ten bits per serial byte (figure 8.1). Each eight-bit message byte specifies either a status value or a data value. The first byte of each message is the status byte, which specifies the type of message and how many additional data bytes follow to make up the complete message. Status bytes always have a most significant bit of 1, whereas data bytes always have a most significant bit of 0. Data bytes thus always range from a minimum value of 0 to a maximum value of 127 (decimal).

Figure 8.1 Transmission protocol of a MIDI byte.



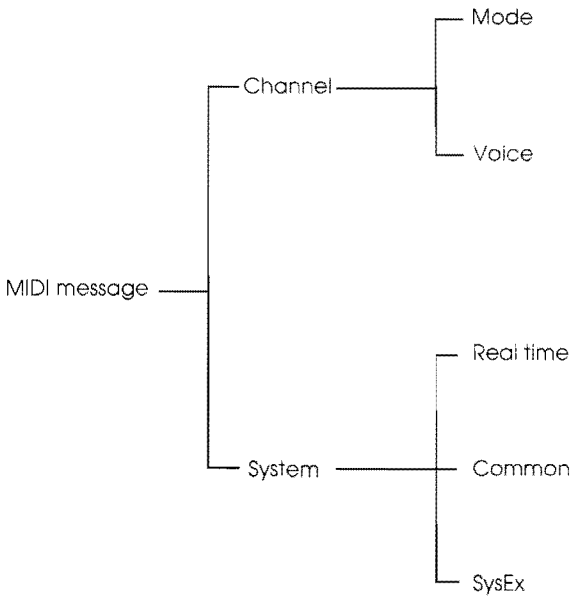
For a summary of status bytes, see table I of the *MIDI 1.0 Detailed Specification,* included in the appendix at the end of this book.

CATEGORIES OF MIDI MESSAGES

MIDI messages may be grouped into two major categories: channel messages and system messages. Channel messages specify a particular MIDI channel within the message and generally control such aspects of performance as Note On/Off, Continuous Controller, and Program Change. System messages affect the MIDI system as a whole without respect for a particular MIDI channel. These include such functions as System Exclusive (SysEx) messages, which provide a way to communicate manufacturer-specific data that are not part of the MIDI specification, and timing clock messages, which are used for synchronization (see chapter 9).

Channel messages may themselves be grouped into channel mode messages and channel voice messages. System messages may be grouped into system real-time and system common messages and the single SysEx message, as shown in figure 8.2.

Figure 8.2 Categories of MIDI messages.



■ CHANNEL MODE MESSAGES

Channel mode messages affect the device that receives them only when they are sent over the channel to which the receiving device is assigned. That assignment of basic channel cannot be set or changed by any MIDI mode or voice message. Rather, basic channel may be set in one of three ways: permanently by the device's manufacturer, from the front-panel controls of the device, or by transmitting an appropriate SysEx message to the device.

Channel mode messages are used primarily to select one of the four MIDI modes described in the previous chapter. Each channel mode message is three bytes long. The four most significant bits of the channel mode status byte are 1011. The four least significant bits designate one of MIDI's sixteen channels, from 0000 for channel 1 to 1111 for channel 16.

The status byte of a channel mode message is identical to the status byte of a Control Change message (described below). The value of the second byte, from 01111001 (121 decimal) to 01111111 (127 decimal), identifies the message as a channel mode message. A value of 01111100 (124 decimal) specifies Omni Off, a value of 01111101 (125 decimal) specifies Omni On, a value of 01111110 (126 decimal) specifies Mono On, and a value of 01111111 (127 decimal) specifies Poly On.

The third byte of a Mono On message specifies the number of channels over which monophonic channel voice messages are to be sent, using the values 1 to 16 (decimal). If the third byte of a Mono On message has a value of 0, the message directs the receiving device to assign its voices one per channel, from its basic channel through channel 16, until all available voices are used.

Three other channel mode messages are also worth noting, though these are not commonly used. All Notes Off turns off any voices turned on via MIDI messages. The MIDI specification does not require that receiving devices respond to All Notes Off, nor does All Notes Off turn off voices played by pressing the keys of a MIDI keyboard. It can be useful for troubleshooting and under certain unusual circumstances when notes cannot otherwise be turned off. The first data byte of All Notes Off is 01111011 (123 decimal), and the second data byte has a value of 0.

Sending a Reset All Controllers message causes the device receiving the message to reset the condition of all its controllers—continuous controllers, switches, pitch bend, and pressure—to what the device considers the optimum initial status. Generally, this will set the value of the Mod Wheel to 0 (no modulation), the pitch bend to center (no pitch bend), and so forth. The first data byte of Reset All Controllers is 01111001 (121 decimal), and the second data byte is 0.

The Local Control message is used to enable or disable the connection between the keyboard and the sound-generating circuitry of a MIDI synthesizer. The value of the first data byte of Local control is 01111010 (122 decimal). If the value of the second data byte is 0, Local Off disables the connection, and keyboard data only to the synthesizer's MIDI Out port. In that case, only incoming MIDI data can control the synthesizer's sound-generating circuitry. On the other hand, if the second data byte is 01111111 (127 decimal), Local On restores normal operation where pressing a key causes a note to sound.

Most keyboard synthesizers provide local control from the front panel, but Local Control messages can also be useful when performing live along with a stored sequence. The sequence can turn the synthesizer's local control on and off at the appropriate times. As the keyboard responds to local control messages in the sequence, it alternates between specifying synthesizer notes and controlling other MIDI equipment in the performance setup.

For a summary of channel mode messages, see Table IV of the *MIDI 1.0 Detailed Specification*, included in the appendix.

CHANNEL VOICE MESSAGES

Channel voice messages specify the nature of sounds produced by the synthesizer that receives the messages, using the channel number embedded in the status byte of the message. The events described by channel voice messages include Note On, Note Off, Channel Pressure, Polyphonic Key Pressure, Program Change, Control Change, and Pitch Bend.

Note On

A Note On message signals the device receiving it to begin playing a note. A MIDI keyboard generates Note On messages when a key is pressed. Other MIDI devices similarly trigger Note On messages.

Like all MIDI messages, the first byte of the Note On message is its status byte. The four most significant bits of the status byte, 1001, identify the message as a Note On and specify that two additional bytes of MIDI data will follow. The four least significant bits of the Note On status byte indicate which channel the message is intended for, from 0000 for channel 1 to 1111 for channel 16.

The second byte of the Note On message identifies which note is called for. Because it is a data byte, its most significant bit is 0. The MIDI specification designates 128 different notes, numbered from 0 to 127, indicated by the remaining seven bits of the second Note On byte. A MIDI note number does not designate a particular pitch. Rather, the note generated depends on the note mapping and tuning determined by the synthesizer or sampler producing the note. Synthesizers most commonly assign MIDI note 60 to middle C and

interpret each higher value as an additional half-step above middle C. Under this typical synthesizer's interpretation of Note On messages, MIDI note 66 would represent the F-sharp above middle C, MIDI note 54 would represent the F-sharp below middle C, and so forth.

The third byte of the Note On message indicates the velocity (usually interpreted as loudness), from 0 to 127, in the same manner as the second Note On byte specifies note number. Just as the note number does not specify a particular pitch, the velocity value does not specify a particular loudness, which instead is determined by the velocity map of the synthesizer or sampler.

Note Off

Each Note Off message signals the end of a note by signaling the device receiving it to turn off one of the notes it is playing. Like Note On, it is a three-byte message: status byte, note data byte, and velocity data byte. The only difference between the two is that the four most significant bits of a Note Off status byte are 1000, whereas a Note On has 1001 in the corresponding positions.

A higher velocity value usually is implemented as a note whose release occurs more quickly; a velocity value of 127 specifies the fastest possible release. Release data generally control how a note dies away; a low velocity value would thus cause the note to die away slowly, and a high velocity value would cause it to die away quickly. Not all MIDI devices implement Note On velocity; fewer still implement Note Off velocity. In such cases, the device transmits a velocity value of 64 for each Note On or Note Off message it transmits or ignores any note off velocity values it receives.

Channel Pressure

Those devices that can sense overall pressure on their keys but are not able to distinguish the pressure put on each individual key send Channel Pressure messages. That is, whichever key is pressed hardest determines the channel pressure value for the entire keyboard. In the same fashion, certain wind controllers transmit channel pressure information on the basis of increasing breath pressure after the initial attack of a note. Depending on the device, it may transmit one or, more commonly, a stream of Channel Pressure messages during the duration of a held note.

How a device receiving a Channel Pressure message responds depends on the particular device. Some devices use channel pressure for aftertouch volume; that is, pressing a key harder once the note has begun sounding causes the note to become louder. Other devices use channel pressure to control vibrato, harder pressure resulting in a faster or wider vibrato. Still other devices use channel pressure to control other parameters or allow you to assign the parameter yourself.

A Channel Pressure message contains two bytes. The status byte has 1101 as its four most significant bits; like all channel messages, the four least significant bits of the status byte specify the channel.

The second byte of the Channel Pressure message specifies a data value from 0 to 127, with 0 indicating minimum pressure and 127 indicating maximum pressure.

The value of a Channel Pressure message affects all the notes currently sounding for that particular MIDI channel provided that the device receiving the message implements a feature that responds to Channel Pressure messages. Otherwise, the receiving device simply ignores incoming Channel Pressure messages.

Polyphonic Key Pressure

Polyphonic key pressure acts in a manner similar to channel pressure except that the Polyphonic Key Pressure message contains an additional data byte. It specifies a particular note and affects only that note rather than every note of the specified channel. A device that transmits Polyphonic Key Pressure messages must be capable of detecting the different pressures put on each of its keys. In addition, the device receiving messages must be able to act on incoming Polyphonic Key Pressure messages if they are to have any effect.

The most significant bits of a Polyphonic Key Pressure message status byte are 1010; the least significant bits specify the channel. Unlike the Channel Pressure message, the second byte of a Polyphonic Key Pressure message specifies a particular note. It uses the values from 0 to 127 to specify a note in the same manner as a Note On message within the channel specified by the status byte. The third byte of the Polyphonic Key Pressure message represents the pressure value, from 0 to 127.

Program Change

The terms *program, patch,* and *voice* refer to a particular setting of a synthesizer's controls that causes the synthesizer to produce a specific instrumental timbre or other musical sound. The terms may also refer to the settings on some other MIDI device such as a digital effects processor that cause the device to produce, for example, a particular reverberation pattern. (*Program,* used in this context, should not be confused with a computer program, a set of instructions that directs a computer to perform a particular task.

A Program Change message does not specify the synthesizer settings themselves. Rather, the synthesizer associates the settings with a patch number, and the Program Change message simply calls up whatever settings the synthesizer associates with that number. The settings that produce a sound vary widely from one synthesizer to another. It is much simpler for MIDI just to specify a program number and leave the particulars of what that program number represents to each synthesizer. (By contrast, SysEx messages described below, can be used to change a synthesizer's parameters governing sound production, but such messages are often hundreds, or even thousands, of bytes long.)

A Program Change message has only a status byte and a single data byte. The status byte of a Program Change message has 1100 as its four most significant bits. As with all channel messages, the four least significant bits of the status byte specify the channel.

The data byte of the Program Change message selects a patch on the device receiving the message, from patch 0 to 127. On receiving a Program Change message, the device receiving the message calls up the patch corresponding to the patch value in the message.

Patch numbers may produce very different results on different synthesizers. Selecting the same patch number on two different synthesizers offers no assurance that the resulting two sounds will be similar. In other words, one synthesizer might associate patch number 47 with a flute sound. Another synthesizer might associate patch number 47 with a trombone. General MIDI, described in chapter 4, provides a minimum level of patch compatibility among synthesizers that implement the GM specification. Even then, patches beyond the 128 specified by GM are likely to differ from one synthesizer to another.

Some synthesizers store fewer than the 128 patches the Program Change message can select. If a synthesizer receives a value higher than the number of stored patches, it may ignore the message, select a default patch, or "wrap" the high value around the number of patches to find a value that is the remainder.

Other synthesizers and MIDI devices can store many more than 128 patches. In such a case, the user can assign 128 of the device's patches to MIDI's 128 patch numbers at any given time. Only those 128 patches (sometimes referred to as the MIDI Program Table) are accessible using Program Change messages.

Even some synthesizers that cannot store more than 128 patches have a user-definable MIDI Program Table. In that way, the user can reconcile two devices that use different patch numbers for similar sounds. For example, a synthesizer might be programmed to map program number 115 to twelve-string acoustic guitar, the same program change number a transmitting synthesizer or sequencer used for acoustic twelve-string guitar.

A special type of Continuous Controller message allows users of synthesizers that implement the Bank Select feature to address more than 128 Program Change numbers by selecting a particular "bank" of 100 programs before selecting a particular Program Change number within that bank. (For more information, see "Bank Select" below.)

Control Change

A Control Change message affects a sound that a Note On message has already initiated, altering such parameters as volume or modulation of the note in progress. The message can specify any of 121 specific MIDI controllers, though most transmitting devices generally implement only a few. These include continuous controllers, On/Off switch controllers, data controllers, and undefined controllers.

Continuous controllers can transmit control data over a range of values, normally 0 to 127 (or 0 to 16,384 in the case of high-resolution controllers, discussed below). On/Off switch controllers transmit only one of two possible values: 0 to represent off or 127 to represent on. Data controllers transmit specific data values directly or increment or decrement the value previously sent. Finally, undefined controllers may operate in the same fashion as continuous controllers, switches, or data controllers, but they use controller numbers that the MIDI specification left unassigned.

Control Change messages consist of three bytes. The four most significant bits of the first (status) byte, 1011, identify the message as a control change. The four least significant bits indicate which channel the message is intended for, from 0000 for channel 1 to 1111 for channel 16.

The second byte specifies a controller number, from 0 through 120 (decimal). As described in the section on channel mode messages above, a second-byte data value of 121 through 127 does not specify a Control Change message at all but rather a channel mode message.

The third byte specifies the setting value for the particular controller specified by the previous byte. Setting values range from 0 to 127, but, as we will see, Control Change messages for a particular continuous controller can be linked together to yield an effective range of 0 to 16,383.

Controller numbers 0 through 31, as specified by the second byte of a Control Change message, are for continuous controllers, that is, those that generate data over a range of values, such as pedals, levers, wheels, and so forth. Some continuous controllers, called bidirectional continuous controllers, are at a centered position while at rest. Using such a controller involves moving it above or below its centered position. On release, a bidirectional continuous controller returns to its at-rest, centered position.

The data byte for each continuous controller uses 0 to represent the minimum position of the controller and 127 to represent the maximum position. In the case of bidirectional continuous controllers, a value of 64 represents the centered position, whereas lower or higher values represent a change from the at-rest position.

If more than 128 steps of resolution is needed, controller numbers 32 through 63 can be used to represent the least significant byte of a two byte value. The corresponding controller number 0 to 31 represents the most significant byte. For example, an instrument might send controller number 7 (volume) to adjust a note's volume from its initial setting (as specified by the velocity value of the note's Note On message). A single Control Change message using controller number 7 can represent any of the 128 settings for that controller. If finer resolution is required, the instrument might also send controller number 39, the corresponding controller number to controller number 7. The seven-bit data value from the controller 7 message represents the most significant byte, whereas the seven-bit data value from the controller 39 message represents the least significant byte. Together, the two messages specify a fourteen-bit data value for the volume setting, providing a resolution of 16,384 steps instead of 128 steps.

Once the most significant byte and corresponding least significant byte of a single Control Change pair have been sent, subsequent fine adjustments to the controller can be made by sending

only the three-byte Control Change message for the least significant byte. The message for the most significant byte does not have to be transmitted again.

By agreement between the MIDI Manufacturers' Association and the Japan MIDI Standards Committee, certain controller numbers are designated for certain standard musical applications. Some of the most important of these include the following: controller number 1, modulation wheel or lever; controller number 2, breath controller; controller number 4, foot controller; controller number 7, main volume; controller number 65, portamento; controller number 66, sostenuto; and controller number 67, soft pedal.

For a summary of controller number assignments, see Table III of the *MIDI 1.0 Detailed Specification,* included in the appendix.

Controllers produce a particular effect, and the third byte of the Control Change message specifies how much effect to produce. In most cases, a value of 0 specifies no effect, and a value of 127 specifies maximum effect.

Three defined MIDI controllers work in a different fashion, however. These are controller number 8, balance; controller number 10, pan; and controller number 11, expression.

The balance controller determines the volume balance between two different sound sources. A controller 8 value of 0 specifies full volume for the left, or lower, half. A value of 64 specifies equal balance, and a value of 127 specifies full volume for the right, or upper, half.

The pan controller determines the location of a single sound source within a stereo sound field. A controller 10 value of 0 specifies hard left, a value of 64 specifies center, and a value of 127 specifies hard right.

Expression is an accented volume above the main volume of a note. Controller 11 values are thus added to the programmed or main volume value.

Although the MIDI specification defines 121 controllers, an instrument manufacturer may choose to implement a controller in a manner different from what the specification might lead you to expect. For example, controller numbers 64 through 69 are assigned functions normally associated with on/off switches, such as sustain and soft pedals. However, manufacturers may choose instead to implement these controller numbers as continuous controllers simply by using the third byte of the Control Change message to specify a range of values from 0 to 127 rather than just 0 (off) and 127 (on).

In the same fashion, a controller number that is normally used for continuous controller data such as controller number 1 (mod wheel)

may be implemented as an on/off modulation switch by sending only third-byte values of 0 (off) and 127 (on). The only certainty of Control Change messages is that a second-byte value represents the most significant byte of a high-resolution controller number 0 to 31, whereas a second-byte value represents the least significant byte of the corresponding controller number 32 to 63.

Pitch Bend

A Pitch Bend Change message transmits a new setting for a pitch bend controller. It is a special case of Control Change message, with a unique status byte value, whose four most significant bits are 1110. Unlike other Control Change messages, Pitch Bend messages are always sent with fourteen-bit resolution. Following the first (status) byte, the second byte of a Pitch Bend message specifies the least significant byte value, and the third byte of the message specifies the most significant byte value. Maximum downward pitch bend corresponds to data byte values of 0, followed by 0. The center (no bend) position corresponds to data byte values of 0, followed by 64. Maximum upward pitch bend corresponds to data byte values of 127, followed by 127.

Sensitivity to Pitch Bend Change messages is a function of the receiving device. That is, two devices that receive the same message specifying maximum downward pitch bend may bend the pitch by different amounts.

For a summary of channel voice messages, see Table II of the *MIDI 1.0 Detailed Specification,* included in the appendix.

Bank Select

MIDI program tables map specific patches to each of the patches accessible through MIDI's 128 program change messages. They are cumbersome to set up and do nothing to make more than 128 patches available at once. Yet most recent synthesizers incorporate many more than 128 patches.

Bank Select, an extension of the MIDI program change message, addresses this problem. Bank Select makes it possible to access any of 16,384 different banks of 128 patches each quickly and easily without setting up MIDI program tables.

Bank Select uses continuous controllers number 0 and 32 (decimal) to communicate the most and least significant bytes, respectively, of a single bank number. Using two data bytes to specify a single bank

number provides for 16,384 individual banks, each containing up to 128 different patches. That is more patches than any current synthesizer implements. However, it is a large enough number that the bank select message should suffice even when tens of thousands of patches might be immediately accessible from optical storage or other high-speed storage devices.

Three separate MIDI messages make up a single bank select command. Two three-byte Control Change messages are followed by a single two-byte Program Change message; each is structured like the Control Change and Program Change messages described above.

The four most significant bits of each bank select Control Change message status byte, 1011, identify the message as a control change. The four least significant bits indicate which channel the message is intended for, from 0000 for channel 1 to 1111 for channel 16. The second byte of the first Control Change message, 00000000, specifies that the Bank Select most significant data byte follows. The second byte of the second Control Change message, 00100000, specifies that the Bank Select least significant data byte follows. Together, the seven data bits of each Control Change message's third-byte data value specify one of the 16,384 possible banks. Finally, the two-byte Program Change message, just as above, specifies a particular patch in the specified bank.

The two Bank Select Control Change messages must be transmitted as a pair before the Program Change message is transmitted. Fourteen-bit Bank Select values correspond to bank numbers as follows:

Most Significant Byte	Least Significant Byte	Bank Number
0000000	0000000	Bank 1
0000000	1111111	Bank 128
0000001	0000000	Bank 129
1111111	1111111	Bank 16,384

A complete Bank Select message on MIDI channel 3 calling up bank 2, program number 7, would appear as follows:

10110010	00000000	00000000	Control Change, channel 3, MSB = 0
10110010	00000010	00000001	Control Change, channel 3, LSB = 1
11000010	00000110		Program Change, channel 3, data value = 6

SYSTEM MESSAGES

Some MIDI messages affect an entire device in a MIDI system or affect every device in the system without regard for any specific MIDI channel. Such system messages are of three types: system common messages, system real-time messages, and a single SysEx message. System common messages are one or two bytes long, system real-time messages are each one byte long, and SysEx messages may be of any length, with a minimum length of three bytes.

The most significant bits of every system message status byte are 1111. Because system messages affect an entire MIDI device rather than just a single channel of that device, system messages contain no channel identifier data. Instead, the four least significant bits of a system message status byte identify the particular type of system message.

System Common Messages

System common messages perform such functions as establishing a common tuning for all devices in a MIDI system or coordinating the selection of a song whose individual parts may be distributed among several MIDI devices. The MIDI specification includes seven system common messages, though only five of these are currently defined: Tune Request, Song Select, Song Position Pointer, End of Exclusive (EOX), and MIDI Time Code Quarter Frame.

Tune Request

Tune Request is a one-byte (status only) message whose value is 11110110 (246 decimal). The message directs all devices receiving it to tune themselves. A MIDI system that includes only digital synthesizers would probably never need to issue a Tune Request. One advantage of digital synthesizers is that they do not drift out of tune if they are working properly.

Some MIDI setups may include analog synthesizers with MIDI retrofits as well as digital synthesizers. Also, certain MIDI manufacturers have continued to make analog synthesizers because some musicians prefer the types of sounds analog synthesizers produce. However, such analog synthesizers use electronic oscillators, which are more likely to have tuning problems during routine use than all-digital synthesizers.

If your MIDI setup includes such analog synthesizers, you may want to issue a Tune Request message between songs. If the master device in a MIDI network is an analog keyboard synthesizer, the keyboard may have a Tune Request switch. Pushing the switch would transmit a Tune Request message down the cable to all subsequent devices in the network, including other analog synthesizers. Alternatively, the message could be stored at the beginning of each recorded sequence and transmitted by the sequencer to all the devices in the setup.

Song Select

The term *song* is used to describe a completed sequence that is ready for playback. A song may consist of a series of drum machine patterns grouped together, a sequence in chorus-and-verse structure stored within a single synthesizer, or a complex, through-composed sequence stored in a personal computer and involving multiple MIDI devices.

Many sequencers and drum machines have the capability to associate particular songs with numbers that identify them, just as synthesizers associate patch settings with program numbers. Just as a MIDI Program Change number can call up a synthesizer patch, a Song Select message can call up a stored song.

Song Select is a two-byte message. The first byte, status, has a value of 11110011 (243 decimal). The value of the second byte, data, can range from 0 to 127, selecting the song associated with that number.

Some receiving devices do not respond to Song Select, in which case the device ignores incoming Song Select messages. Furthermore, just as patches on two different synthesizers are unlikely to correspond, there is no requirement that songs be numbered consistently from one device to another. It is the responsibility of the user to ensure that the correct song has been mapped to the appropriate song number within each device that will respond to incoming Song Select messages.

Song Position Pointer

The MIDI specification states that each musical duration represented by a quarter note be divided into twenty-four "ticks" of the MIDI clock (also called timing clock). Other musical durations can similarly be expressed in terms of their MIDI clock lengths: for example, one half note equals forty-eight ticks, and one sixteenth note equals six ticks.

A MIDI clock tick does not represent a specific elapsed time. Rather, the length of a musical duration such as a quarter note is a function of tempo. At a faster tempo, a quarter note represents a shorter elapsed time than the same quarter note at a slower tempo. MIDI clock ticks thus form the basis for establishing musical durations and synchronization, not elapsed time.

Most MIDI sequencers and drum machines can count MIDI beats from the beginning of a song. Six ticks comprise a single MIDI beat. Because six ticks of the MIDI clock correspond to one sixteenth note, a MIDI beat is equivalent to a sixteenth note. A sequencer's Song Position, then, is the number of elapsed MIDI beats since the start of the song, ranging from 0 to a maximum of 16,383 MIDI beats. (Bear in mind that a MIDI beat is not equivalent to a musical beat, which varies with the time signature of each piece of music.)

To further complicate matters, a particular sequencer may implement greater precision than MIDI's twenty-four ticks per quarter note. Typically, the sequencer's time base would be a multiple of the MIDI clock, that is, forty-eight ticks per quarter note, ninety-six ticks per quarter note, or some similar multiple.

The Song Position Pointer message consists of a status byte followed by two data bytes that specify Song Position. The status byte has a value of 11110010 (242 decimal). It is followed first by the least significant byte of song position data, then the most significant data byte. Because each data byte is a seven-bit value, the two bytes combined have 14-bit precision, representing the 16,384 possible song position values.

Activating a master sequencer's start control normally sets song position to 0. All other sequencers that derive their timing clocks from the master would then also begin playing their sequences from the beginning. As the combined sequences play, Song Position is automatically incremented every sixth MIDI clock tick until the master sequencer's stop control is activated. If the master sequencer's continue control is activated, song position is again incremented from its current value.

Song Position Pointer messages are effective only if they are sent before a sequencer begins to play a song; they cannot be used to synchronize devices to a song that is already playing. On receipt of a Song Position Pointer message, every device set to MIDI sync (external) clock mode sets its own song position pointer to that of the incoming Song Position Pointer message.

For example, if the value of song position is 20 at the time a Song Position Pointer message is transmitted and a device receiving the

Song Position Pointer message uses a resolution of 96 internal clock ticks per quarter note, that device would set its internal clock as follows:

1. Song Position (20) × 6 MIDI clock ticks per MIDI beat = 120 MIDI clock ticks
2. 120 MIDI clock ticks × (96 internal clock ticks per quarter note / MIDI's time base of 24 clock ticks per quarter note) = 480 internal clock ticks

The device receiving the Song Position Pointer message would therefore set its internal clock to begin playback 480 ticks into the sequence.

Song Position Pointer messages provide only the most basic of synchronization capabilities, namely, the capability to start two clock-sensitive devices at some point other than the beginning of a sequence. Furthermore, the MIDI specification does not require MIDI devices to implement Song Position Pointer, and few manufacturers have implemented it. Other MIDI messages related to Song Position Pointer are grouped under system real time and will be discussed below. (For more on synchronization, see chapter 9.)

End of Exclusive

A SysEx message may contain any number of bytes, so the End of Exclusive (EOX) message alerts devices receiving the SysEx message that it is the last byte of that SysEx message. End of Exclusive is a single-byte message whose value is 11110111 (247 decimal). Although the MIDI specification classifies EOX as a system common message, it functions only in connection with a SysEx message, so it will be discussed further in the section below covering SysEx messages.

MIDI Time Code Quarter Frame

MIDI Time Code is a synchronization system and will be covered in chapter 9. MIDI Time Code Quarter Frame is a two-byte message whose status byte value is 11110001 (241 decimal). It specifies a high-resolution timing value for other devices to synchronize to.

For a summary of system common messages, see Table V of the *MIDI 1.0 Detailed Specification,* included in the appendix at the end of this book.

System Real-Time Messages

As the name implies, system real-time messages coordinate and synchronize the timing of clock-based MIDI devices such as sequencers, drum machines, and synthesizers during performance or playback. Consequently, such messages must be sent at regular, precise intervals to ensure that every device in a MIDI system marches to the same beat. System real-time messages are so important that they are sometimes inserted between bytes of other multibyte messages.

Many MIDI devices have no clock or synchronization capabilities, and such devices simply ignore incoming system real-time messages.

All system real-time messages consist of only a single status byte without any subsequent data bytes. Because system real-time messages affect an entire MIDI device rather than just a single channel of that device, they, like all system messages, contain no channel identifier data.

Timing Clock

The one-byte value of a Timing Clock message is 11111000 (248 decimal). When a master sequencer plays a song, it transmits Timing Clock messages at a rate of twenty-four per quarter note, interspersed among the other MIDI messages it transmits. The master (transmitting) sequencer is thus able to transmit tempo and duration information by adjusting the rate at which it sends Timing Clock messages according to the tempo and rhythmic values. Any clock-based MIDI systems set to MIDI (external) clock mode synchronize to the incoming Timing Clock messages, providing a means to synchronize the entire system using MIDI messages. To avoid confusion, only one device in a MIDI system should serve as the master clock at any given time. All other clock-based devices in the system should be set to receive timing information from the external master clock.

Start

Activating the Play control on a master timing device sends a Start message. It is a one-byte, status-only message whose value is 11111010 (250 decimal). It directs any receiving devices that respond to incoming system real-time messages to start play from the beginning of their stored song or sequence. Timing Clock messages then serve to initiate and maintain synchronization from that point.

Those devices that implement Song Position Pointer also return their Song Position value to 0 on receipt of a Start message.

Stop

The one-byte. status-only Stop message is sent when playback is stopped on the device that supplies master Timing Clock information to the rest of the system. Its value is 11111100 (252 decimal). Any receiving devices that respond to incoming system real-time messages will cease playback immediately on receiving a Stop message. Song Position within each of the devices does not change from its current value as of when it transmitted or received the Stop message.

Continue

Continue—a one-byte, status-only message whose value is 11111011 (251 decimal)—initiates playback on any receiving devices that respond to incoming system real-time messages in the same manner as the Start command. The difference is that Continue initiates playback starting with the current value of Song Position at the time the message is received. A Start message initiates playback from the beginning of the song or sequence and resets Song Position to 0.

Some transmitting devices have a separate Continue control. Others send a Start message if the Song Position value of the transmitting device is 0 and a Continue message if the value is greater than 0. If playback on a transmitting device begins and stops before receiving devices have been set to external synchronization, the receivers' Song Position values might not correspond to the master's. In that case, they might receive a Continue message generated by activating the master device's Play control yet not play back in proper synchronization to the master.

The example above is one illustration of the limitations of MIDI sync. It cannot provide true chase-lock synchronization, in which receiving devices automatically synchronize correctly to the transmitting device regardless of the point at which the transmitting device begins playback.

Despite their limitations, Stop, Start, and Continue provide a way to stop playback and then resume without having to begin once again from the beginning.

Active Sensing

Active Sensing, like System Reset (below), solves an infrequently encountered problem; as a result, few MIDI devices implement it. The single-byte, status-only Active Sensing message has a value of 11111110 (254 decimal). Transmitting devices that implement Active Sensing send it at least as often as every 300 milliseconds whenever they are transmitting no other MIDI data.

Once a device that implements active sensing receives an Active Sensing message, that device will continue to look for Active Sensing messages at regular intervals and at those times when there are no other incoming MIDI messages. If it fails to receive the expected Active Sensing messages and no other MIDI messages appear along the incoming MIDI cable, it would likely be because the cable connecting the two devices had become unplugged.

An unexpected cable disconnection can cause a variety of problems. At best, the receiving device will simply fail to respond to subsequent messages from the transmitting device. At worst, the MIDI cable might become unplugged after the transmitting device sent a Note On message but before the transmitter sent its companion Note Off message. In that case, the receiving device would continue to sustain its note indefinitely—the dreaded "stuck," or "hung," note.

Although implementing active sensing can avoid the problems described above, it creates problems of its own. Sending repeated messages at frequent intervals contributes to MIDI choke, the overloading of the MIDI data stream described in chapter 7. MIDI choke is more likely to become a problem under normal circumstances than disconnected cables, so implementing active sensing carries a high price.

Choosing not to implement active sensing causes no compatibility problems, so most manufacturers omit it. A receiving device that does implement active sensing must first receive an Active Sensing message from the transmitter before it starts looking for subsequent Active Sensing messages. If the transmitting device does not implement active sensing, the receiving device will not initiate active sensing. In the same manner, if a receiving device does not implement active sensing, it ignores any Active Sensing messages it receives.

System Reset

The one-byte, status-only System Reset message has a value of 11111111 (255 decimal). On receiving a System Reset message, the device receiving the message should return all its controls to their default settings, that is, the initial values of the controls at power up.

Those default settings may vary from one device to another, so a system reset can have different effects on different devices. As a general guideline, the MIDI specification lists the following as its system reset conditions:

1. MIDI Mode 1—Omni On/Poly
2. Local On
3. All voices off (no notes playing)
4. Reset all controllers to minimum or centered position
5. Stop playback of any sequencer or song currently playing
6. Clear Running Status (see below for an explanation of Running Status)
7. Reset other parameters to their values at power on

The MIDI specification further suggests that System Reset be used sparingly and be transmitted under manual control rather than as a stored sequencer message. It should generally be used only as a last resort to return all devices to a known condition, not as part of an initialization routine. Many MIDI devices retain their most recent settings on power down so that they maintain the same settings on the following power up. This capability can be a time-saver when a session must be interrupted and continued later, but those settings might be lost if a System Reset is performed.

For a summary of system real-time messages, see Table VI of the *MIDI 1.0 Detailed Specification,* included in the appendix at the end of this book.

System Exclusive Messages

The designers of the MIDI specification recognized that they could not include messages to address the unique needs of each MIDI device and that it would be unwise to try. Instead, they provided System Exclusive (SysEx) messages, which communicate device-specific data that fall outside the scope of standard MIDI messages.

MIDI devices made by a single manufacturer can use SysEx messages to transmit and receive data that are unique to that manufacturer's devices. Indeed, any manufacturer may use the system exclusive codes of any existing product without the permission of the original manufacturer; one manufacturer, then, could use SysEx messages to transmit data to a different manufacturer's device. Whether between devices from the same or different manufacturers, such data might include synthesizer patch parameters or sampler data values.

If SysEx messages are to be useful, they must represent data that are compatible with the transmitting and receiving device. For example, a SysEx message containing parameter data for an FM synthesizer will have no corresponding meaning to an additive synthesizer even if the same manufacturer made both.

Each SysEx message begins with a status byte whose value is 11110000 (240 decimal). The first data byte following a SysEx status byte identifies a specific manufacturer, using an ID number that must be obtained from the MIDI Manufacturers Association or the Japan MIDI Standards Committee. Such ID values range from 0 to 127: for example, ID number 1 specifies devices manufactured by Sequential, whereas ID number 67 specifies devices manufactured by Yamaha. An ID whose value is 0 specifies that the following two bytes are used as extensions to the manufacturer's ID. For example, the three-byte ID 0, 0, 7 specifies Digital Music Corporation, whereas the three byte ID 0, 0, 16 specifies DOD Electronics. (All the above IDs are in decimal notation.)

Three ID numbers are reserved for specific purposes: 01111101 (125 decimal) is reserved for noncommercial use and may not be used by any product released to the public, 01111110 (126 decimal) identifies a non–real-time SysEx message, and 01111111 (127 decimal) identifies a real-time SysEx message.

Both real-time and non–real-time SysEx messages use the same format. The status byte 11110000 is followed by the ID code data byte, followed by a channel number byte, a sub-ID #1 byte, a sub-ID #2 byte, any number of data bytes, and the EOX message byte 11110111 to conclude the real-time or non–real-time SysEx message.

For a list of currently defined system exclusive sub-ID numbers, see Table VIII of the *MIDI 1.0 Detailed Specification,* included in the appendix at the end of this book. (For a list of manufacturers' SysEx ID numbers, contact the International MIDI Association.)

Once a receiving device determines that the manufacturer's ID in the incoming message is the same as its own, it attempts to act on the incoming SysEx message. If the incoming SysEx message specifies a manufacturer other than the maker of the receiving device, the device ignores the rest of the incoming SysEx message.

Any number of data bytes may follow the manufacturer ID, provided that each is a true MIDI data byte; that is, the most significant bit of each data byte must be 0. The SysEx message ends with an EOX message, the one-byte, status-only system common message whose value is 11110111 (247 decimal).

Once a SysEx message has begun, no other MIDI status or data bytes should be sent until an EOX has terminated the SysEx message.

The only exception is system real-time messages. They may be inserted between bytes of any other MIDI message and are received even by devices that are ignoring the SysEx message bytes the system real-time messages are interleaved between. However, receiving a status byte other than real time or EOX during a SysEx message transmission terminates receipt of the SysEx message. Any SysEx data bytes that follow are ignored.

Two SysEx messages control GM synthesizers that can operate in either GM mode or a mode specific to that synthesizer that does not comply with GM. One of the messages turns on GM mode in such synthesizers, and the other turns GM mode off.

The first message, to turn GM mode on, begins with the SysEx status byte, 11110000 (240 decimal). The first data byte, 01111110 (126 decimal), identifies the message as a non–real-time message. The following data bytes specify the manufacturer's ID number of the target device, as described above. The sub-ID #1 byte, 00001001 (9 decimal), identifies the message as a GM message, and the sub-ID #2 byte, 00000001 (1 decimal), identifies the message as GM On. The message ends with the one-byte, status-only EOX system common message, whose value is 11110111 (247 decimal).

The only difference between the Turn GM Off message and the Turn GM On message above is that the sub-ID #2 byte of the Turn GM Off message is 00000010 (2 decimal).

To turn on GM mode on for all devices connected to the system, use the Broadcast device ID, 01111111 (127 decimal), rather than a specific manufacturer's ID. The MIDI specification includes the Broadcast ID number so that a single SysEx message can address every device, as in this situation.

The pair of messages, then, is as follows:

Turn GM System On

11110000	System Exclusive status byte
01111110	non–real-time SysEx header
01111111	target device: Broadcast
00001001	sub-ID #1: GM Message
00000001	sub-ID #2: GM On
11110111	End of Exclusive (EOX)

Turn GM System Off

11110000	System Exclusive status byte
01111110	non–real-time SysEx header
01111111	target device: Broadcast
00001001	sub-ID #1: GM Message
00000010	sub-ID #2: GM Off
11110111	End of Exclusive (EOX)

For a summary of SysEx messages, see Table VII of the *MIDI 1.0 Detailed Specification,* included in the appendix.

RUNNING STATUS

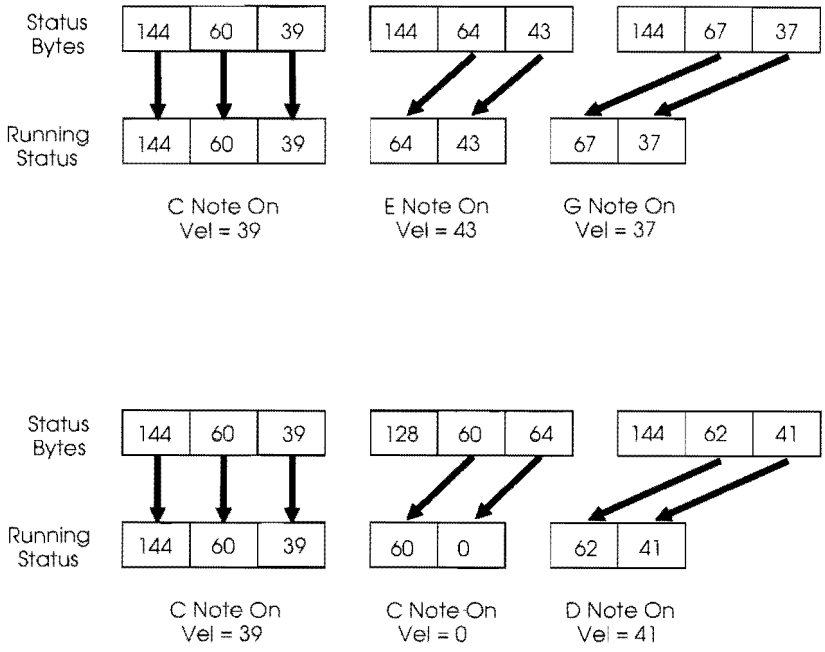
Running status is not a type of MIDI message. It is a shortcut technique of transmission that minimizes message traffic volume and helps avoid the problems that arise from an overloaded MIDI data stream. Under running status, once a message of a particular type has been sent, subsequent messages of the same type that follow immediately after the first are sent without repeating the status byte for each subsequent message.

For example, a series of notes would normally be transmitted as follows: Note On status, Note Number data value, Velocity data value; Note On status, Note Number data value, Velocity data value; Note On status, Note Number data value, Velocity data value; and so forth for each note in the group. Under running status, the same series of notes would be transmitted as follows: Note On status, Note Number data value, Velocity data value; Note Number data value, Velocity data value; Note Number data value, Velocity data value; and so forth. As long as the subsequent data consist of Note On messages using the same MIDI channel as the first complete Note On message, the status bytes of the subsequent messages need not be transmitted. Once a device receives a status byte, it must interpret subsequent data bytes as the same status until it receives a new status byte. In other words, it must maintain the most recent, or "running," status.

Running status is most appropriate for transmitting Note On messages and continuous controller data. In the case of Note On messages, running status can take advantage of the fact that transmitting Note On messages with velocity values of 0 can turn off the corresponding notes. Note Off message need not be transmitted.

Figure 8.3 shows the savings achieved using running status. The first example shows a series of three running status Note On messages as compared to normal Note On transmission. The second shows three Note On messages (the second of which has a velocity value of 0) instead of Note On/Note Off/Note On messages.

Figure 8.3 Status bytes versus Running Status.



A FINAL WARNING

Even if you know and understand all the messages that pass back and forth between MIDI devices, it is easy to overlook this basic principle of MIDI message transmission: Few MIDI devices implement every variety of MIDI message. The MIDI specification does not require that MIDI devices implement any but a very few core features. Unless a device implements a particular feature, it will not respond to incoming messages that call on that feature. Even if the device does implement a feature, how it implements the feature may be different from how another device implements it. The only way to be sure is to understand the messages entering a device and observe how the device responds.