MIDI

INTRODUCTION TO MIDI

Simply stated, Musical Instrument Digital Interface (MIDI) is a digital communications language and compatible specification that allows multiple hardware and software electronic instruments, performance controllers, computers, and other related devices to communicate with each other over a connected network. MIDI is used to translate performance- or control-related events (such as playing a keyboard, selecting a patch number, varying a modulation wheel, triggering a staged visual effect, etc.) into equivalent digital messages and then transmit these messages to other MIDI devices where they can be used to control sound generators and other performance parameters. The beauty of MIDI is that its data can be easily recorded into a hardware device or software program (known as a sequencer), where it can be edited and transmitted to electronic instruments or other devices to create music or control any number of parameters.

In artistic terms, this digital language is an important medium that lets artists express themselves with a degree of flexibility and control that wasn't possible at an individual level beforehand. Through the use of this performance language, an electronic musician can create and develop a song or composition in a practical, flexible, affordable, and fun production environment.

The word *interface* refers to the actual data communications link and software/hardware systems in a connected MIDI network. Through MIDI, it's possible for all of the electronic instruments and devices within a network to communicate real-time performance and control-related MIDI data messages throughout a system to multiple instruments and devices via MIDI, USB, or FireWire networked data lines. Given that MIDI data can simultaneously transmit performance and control messages over multiple channels (usually in groupings of 16 channels per port), an electronic musician can record, overdub, mix, and play back their performances in a building-block fashion that resembles the multitrack recording process. In fact, the true power of MIDI lies in its ability to edit, control, alter and automate parts of a composition after the original performance has been recorded, allowing performance parameters to be easily altered in ways that are unique to the medium.

What MIDI isn't

For starters, let's dispel one of MIDI's greatest myths: MIDI doesn't communicate audio—it cannot create sounds! It is a digital language protocol that can only be used to trigger and/or control a device (which, in turn generates, reproduces, or controls the sound). Thus, the MIDI data and the audio routing paths are kept entirely separate from each another, Figure 16.1. Even if they digitally share the same transmission cable (such as through USB or FireWire), the actual data paths and formats are distinct.

In short, MIDI's control-related language can be thought of as the dots on a player-piano roll—when we put the paper roll up to our ears, we hear nothing. However, when the cutout dots pass over the sensors on a player piano, the instrument itself begins to make beautiful music. The analogy is

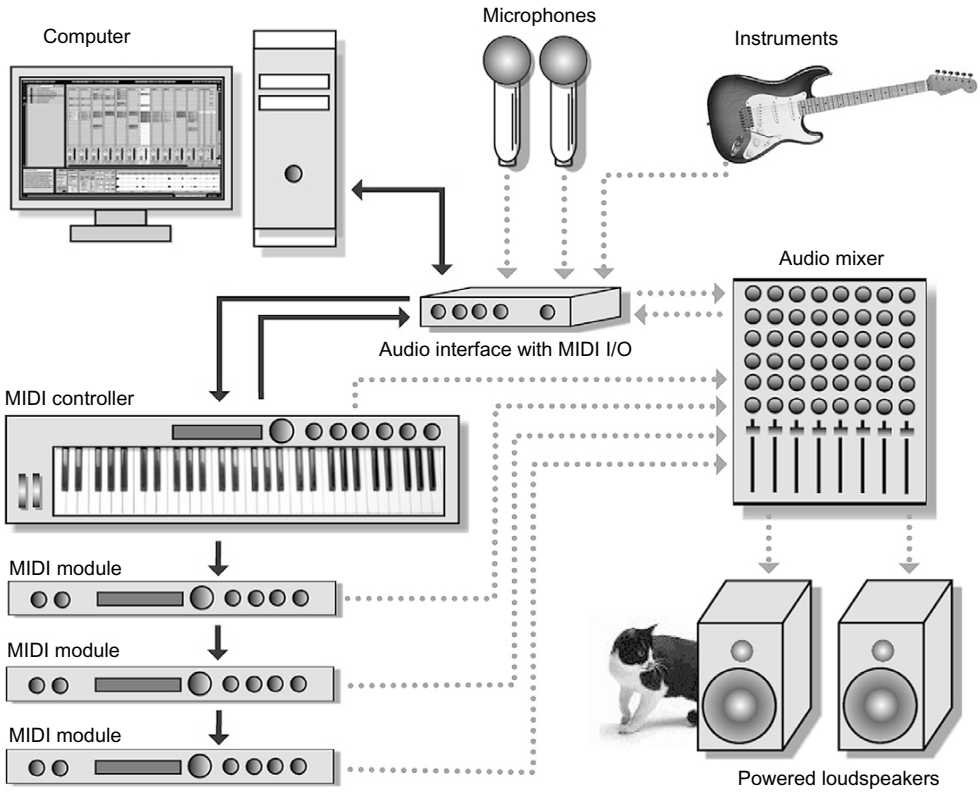


FIGURE 16.1
Example of a typical MIDI system with the MIDI network connections being shown in solid lines and audio connections shown using dotted lines.

pretty much the same with MIDI. A MIDI file or data stream is simply a set of instructions that pass through wires in a serial fashion, but when an electronic instrument interprets the data, we then hear sound.

As a performance-based control language, MIDI complements modern music production, by allowing a performance track to be edited, layered, altered, spindled, mutilated, and improved with relative ease under completely automated computer control and after the fact, during post-production. If you played a bad note, fix it. If you want to change the key or tempo of a piece, change it. If you want to change the expressive volume of a phrase in a song, just do it! Even its sonic character (timbre) can be changed! These capabilities merely hint at the power of this medium that widely affects the project studio, professional studio, audio or visual and film, live performance, multimedia, and even your cell phone!

THE MIDI MESSAGE

From its inception in the early 80s, the MIDI 1.0 spec (which is still the adopted version to this day) must be strictly adhered to by those who design and manufacture MIDI-equipped instruments and devices. As such, users needn't worry about whether the MIDI Out of one device will be understood by the MIDI In of a device that's made by another manufacturer (at least the basic performance level). We need only consider the day-to-day dealings that go hand-in-hand with using electronic instruments, without having to be concerned with data compatibility between devices.

MIDI messages are communicated through a standard MIDI line in a serial fashion at a speed of 31,250 bits/s. These messages are made up of groups of 8-bit words (known as bytes), which are used to convey instructions to one or all MIDI devices within a system. Only two types of bytes are defined by the MIDI specification: the status byte and the data byte.

Table 16.1 Status and Data Byte Interpretation.

	Status Byte	Data Byte 1	Data Byte 2
Description	Status/Channel #	Note #	Attack Velocity
Binary Data	(1001.0100)	(0100.0000)	(0101.1001)
Numeric Value	(Note On/Ch #5)	(64)	(89)
0000 = CH#1	0100 = CH#5	1000 = CH#9	1100 = CH#13
0001 = CH#2	0101 = CH#6	1001 = CH#10	1101 = CH#14
0010 = CH#3	0110 = CH#7	1010 = CH#11	1110 = CH#15
0011 = CH#4	0111 = CH#8	1011 = CH#12	1111 = CH#16

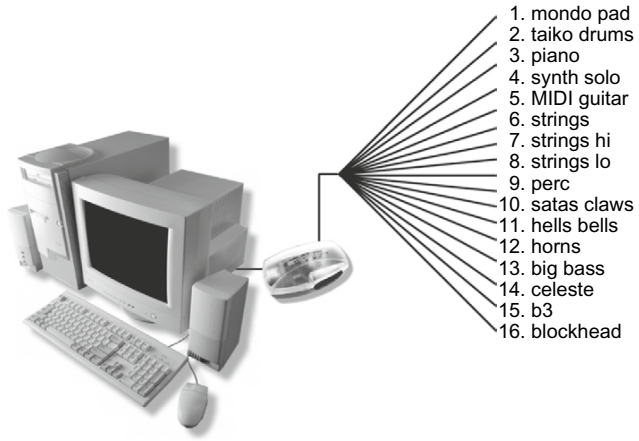


FIGURE 16.2
Up to 16 channels can be transmitted through a single MIDI cable.

A status byte is used to identify what type of MIDI function is to be performed by a device or program. It's also used to encode channel data (allowing the instruction to be received by a device that's set to respond to a specific channel). A data byte is used to associate a value to the event that's given by the accompanying status byte.

The most significant bit (MSB), the leftmost binary bit within a digital word within a MIDI byte, is used solely to identify the data's particular function. The MSB of a status byte is always 1, while the MSB of a data byte is always 0. For example, a 3 byte MIDI note-on message (which is used to signal the beginning of a MIDI note) in binary form might read as shown in Table 16.1. Thus, a 3 byte note-on message of (10010100) (01000000) (01011001) will transmit instructions that would be read as "Transmitting a note-on message over MIDI channel #5, using keynote #64, with an attack velocity (volume level of a note) of 89."

MIDI channels

Just as a public speaker might single out and communicate a message to one individual in a crowd, MIDI messages can be directed to communicate information to a specific device or series of devices within a MIDI system. This is done by imbedding a channel-related nibble (4 bits) within the status byte, allowing data to be conveyed to any of 16 channels over a single MIDI data cable line, Figure 16.2. This makes it possible for performance or control information to be communicated to a specific device or a sound generator within a device that's assigned to a particular channel.

Whenever a MIDI device, sound generator, or program function is instructed to respond to a specific channel number, it will only respond to messages that are transmitted on that channel (i.e., it ignores

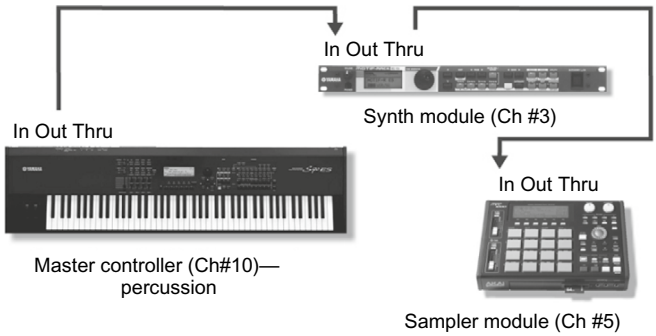


FIGURE 16.3
MIDI setup showing a set of MIDI channel assignments.

channel messages that are transmitted on any other channel). For example, let's assume that we're going to create a short song using a synthesizer that has a built-in sequencer (a device or program that's capable of recording, editing, and playing back MIDI data) and two other synths, Figure 16.3.

1. We could start off by recording a drum track into the master synth using channel 10 (many synths are pre-assigned to output drum/percussion sounds on this channel).
2. Once recorded, the sequence will then transmit the notes and data over channel 10, allowing the synth's percussion section to be heard.
3. Next, we could set a synth module to channel 3, and instruct the master synth to transmit on the same channel (since the synth module is set to respond to data on channel 3, its generators will sound whenever the master keyboard is played). We can now begin recording a melody line into the sequencer's next track.
4. Playing back the sequence will then transmit data to both the master synth (percussion section) and the module (melody line) over their respective channels. At this point, our song is beginning to take shape.
5. Now, we can set a sampler (or other instrument type) to respond to channel 5, and instruct the master synth to transmit on the same channel, allowing us to further embellish the song.
6. Now that the song's complete, the sequencer can then play the musical parts to the synths on their respective MIDI channels, all in an environment that allows us to have complete control of volume, edit, and a wide range of functions over each instrument. In short, we've created a true multichannel working environment.

It goes without saying that the above example is just but one of the infinite setup and channel possibilities that can be encountered in a production environment. It's often true, however, that even the most complex MIDI and production rooms will have a system, a basic channel and overall layout that makes the day-to-day operation of making music easier. This layout and the basic decisions in your own room are, of course, up to you. Streamlining a system to work both efficiently and easily will come over time with experience and practice.

MIDI modes

Electronic instruments often vary in the number of sounds and/or notes that can be simultaneously produced by their internal sound-generating circuitry. For example, certain instruments can only produce one note at a single time (known as a *monophonic* instrument), while others can generate 16, 32, and even 64 notes at once (these are known as *polyphonic* instruments). The latter type is easily capable of playing chords and/or more than one musical line on a single instrument.

In addition, some instruments are only capable of producing a single generated sound patch (often referred to as a voice) at any one time. Its generating circuitry could be polyphonic, allowing the player to lay down chords and bass/melody lines), but it can only produce these notes using a single characteristic sound at any one time (e.g., an electric piano, or a synth bass, or a string patch). However, the vast majority of newer synths differ from this in that they're multitimbral in nature, meaning that they can generate numerous sound patches at any one time (e.g., an electric piano, and a synth bass, and a string patch). That is, it's common to run across electronic instruments that can simultaneously

generate a number of voices, each offering its own control over parameters (such as volume, panning, modulation, etc.) and—best of all—it's also common for different sounds to be assigned to their own MIDI channels, allowing multiple patches to be internally mixed within the device (often top a stereo output bus), or to independent outputs.

As a result of these differences between instruments and devices, a defined set of guidelines (known as *MIDI reception modes*) has been specified that allows a MIDI instrument to transmit or respond to MIDI channel messages in several ways. For example, one instrument might be programmed to respond to all 16 MIDI channels at one time, while another might be polyphonic in nature, with each voice being programmed to respond to only a single MIDI channel.

POLY/MONO

An instrument or device can be set to respond to MIDI data in either the poly mode or the mono mode. Stated simply, an instrument that's set to respond to MIDI data polyphonically will be able to play more than one note at a time. Conversely, an instrument that's set to respond to MIDI data monophonically will only be able to play a single note at any one time.

OMNI ON/OFF

Omni on/off refers to how a MIDI instrument will respond to MIDI messages at its input. When Omni is turned on, the MIDI device will respond to all channel messages that are being received regardless of its MIDI channel assignment. When Omni is turned off, the device will only respond to a single MIDI channel or set of assigned channels (in the case of a multitimbral instrument).

The following list and figures explain the four modes that are supported by the MIDI spec in more detail.

- **Mode 1—Omni On/Poly:** In this mode, an instrument will respond to data that's being received on any MIDI channel, and then redirect this data to the instrument's base channel, Figure 16.2a. In essence, the device will play back everything that's presented at its input in a polyphonic fashion... regardless of the incoming channel designations. As you might guess, this mode is rarely used.
- **Mode 2—Omni On/Mono:** As in Mode 1, an instrument will respond to all data that's being received at its input, without regard to channel designations. However, this device will only be able to play one note at a time, Figure 16.2b. Mode 2 is used even more rarely than Mode 1, as the device can't discriminate channel designations and can only play one note at a time.
- **Mode 3—Omni Off/Poly:** In this mode, an instrument will only respond to data that matches its assigned base channel in a polyphonic fashion, Figure 16.2c. Data that is assigned to any other channel will be ignored. This mode is by far the most commonly used because it allows the voices within a multitimbral instrument to be individually controlled by messages that are being received on different MIDI channels. For example, each of the 16 channels in a MIDI line could be used to independently play each of the parts in a 16-voice, multitimbral synth.
- **Mode 4—Omni Off/Mono:** As with Mode 3, an instrument will be able to respond to performance data that's transmitted over a single dedicated channel; however, each voice will only be able to generate one MIDI note at a time, Figure 16.2d. A practical example of this mode is often used in MIDI guitar systems, where MIDI data is monophonically transmitted over six consecutive channels (one channel/voice per string).

Channel messages

Channel-voice messages are used to transmit real-time performance data throughout a connected MIDI system. They're generated whenever a MIDI instrument's controller is played, selected, or varied by the performer. Examples of such control changes could be the playing of a keyboard, pressing of program selection buttons, or movement of modulation or pitch wheels. Each channel-voice message contains a MIDI channel number within its status byte, meaning that only devices that are assigned to the same channel number will respond to these commands. There are seven channel-voice message types: note-on, note-off, polyphonic-key pressure, channel pressure, program change, pitch-bend change and control change.

Note-On Messages. A note-on message is used to indicate the beginning of a MIDI note. It is generated each time a note is triggered on a keyboard, controller, or other MIDI instrument (i.e., by pressing a key, hitting a drum pad, or by playing a sequence).

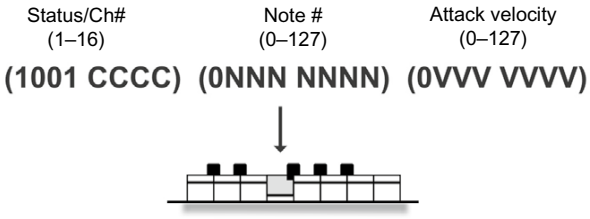


FIGURE 16.4
Byte structure of a MIDI note-on message.

A note-on message consists of 3 bytes of information, Figure 16.4: Note-on status/MIDI channel number, MIDI pitch number and attack velocity value.

The first byte in the message specifies a note-on event and a MIDI channel (1–16). The second byte is used to specify which of the possible 128 notes (numbered 0–127) will be sounded by an instrument. In general, MIDI note number 60 is assigned to the middle C key of an equally tempered keyboard, while notes 21 to 108 correspond to the 88 keys of an extended keyboard controller. The final byte is used to indicate the velocity or speed at which the key was pressed (over a value range that varies from 0 to 127). Velocity is used to denote the loudness of a sounding note, which increases in volume with higher velocity values (although velocity can also be programmed to work in conjunction with other parameters such as expression, control over timbre, sample voice assignments, etc).

Note-Off Messages. A note-off message is used as a command to stop playing a specific MIDI note. Each note-on message will continue to play until a corresponding note-off message for that note has been received. In this way, the bare basics of a musical composition can be encoded as a series of MIDI note-on and note-off events. It should also be pointed out that a note-off message wouldn't cut off a sound; it'll merely stop playing it. If the patch being played has a release (or final decay) slope, it will begin this stage upon receiving the message.

A note-off message consists of three bytes of information, Figure 16.5: Note-off status/MIDI channel number, MIDI pitch number and attack velocity value.

In contrast to the dynamics of attack velocity, the release velocity value (0–127) indicates the velocity or speed at which the key was released. A low value indicates that the key was released very slowly, whereas a high value shows that the key was released quickly. Although not all instruments generate or respond to MIDI's release velocity feature, instruments that are capable of responding to these values can be programmed to vary a note's speed of decay, often reducing the signal's decay time as the release velocity value is increased.

A note-on message that contains an attack velocity of 0 (zero) is generally equivalent to the transmission of a note-off message. This common implementation tells the device to silence a currently sounding note by playing it with a velocity (volume) level of 0.

All Notes Off. On the odd occasion (often when you least expect it), a MIDI note can get stuck! This can happen when data drops out or a cable gets disconnected, creating a situation where a note receives a note-on message, but not a note-off message, resulting in a note that continues to

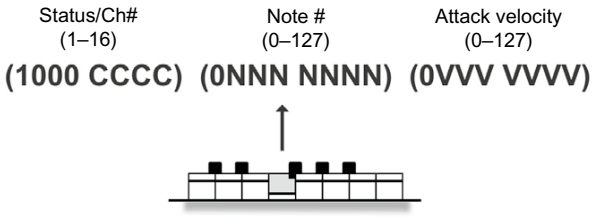


FIGURE 16.5
Byte structure of a MIDI note-off message.

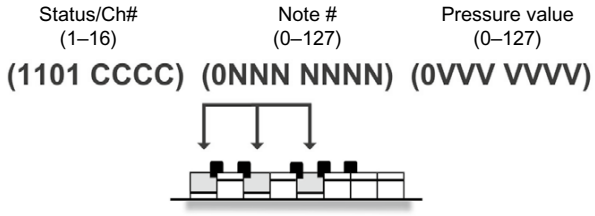


FIGURE 16.6
Byte structure of a MIDI channel-pressure message.

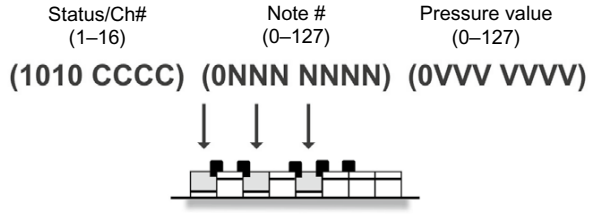


FIGURE 16.7
Byte structure of a MIDI polyphonic-key pressure message.

plaaaaaaaaaaayyyyyyyyyy! Since you're often too annoyed or under pressure to take the time to track down which note is the offending sucka… it's generally far easier to transmit an all notes off message that silences everything on all channels and ports. If it exists, this can easily be done by pressing a Panic Button that's built into the sequencer or hardware MIDI interface.

Pressure (Aftertouch) Messages. Pressure-related messages (often referred to as aftertouch) occur after you've pressed a key and then decide to press down harder to gain a particular effect. For devices that can respond to (and therefore generally transmit) these messages, aftertouch can often be assigned to such parameters as vibrato, loudness, filter cut-off, and pitch. Two types of pressure messages are defined by the MIDI spec:

- Channel-pressure.
- Polyphonic-pey pressure.

Channel-pressure messages are commonly transmitted by instruments that only respond to a single overall pressure, regardless of the number of keys that are being played at any one time, Figure 16.6. For example, if six notes are played on a keyboard and additional aftertouch pressure is applied to just one key, the assigned parameter would be applied to all six notes.

A channel-pressure message consists of 3 bytes of information, Figure 16.6: Channel-pressure status/MIDI channel number, MIDI note number, and pressure value.

Polyphonic-key pressure messages respond to pressure changes that are applied to the individual keys of a keyboard. That's to say that a suitably equipped instrument can transmit or respond to individual pressure messages for each key that's depressed.

How a device responds to these messages will often vary from manufacturer to manufacturer (or can be assigned by the user). However, pressure values are commonly assigned to such performance parameters as vibrato, loudness, timbre, and pitch. Although controllers that are capable of producing polyphonic pressure are generally more expensive, it's not uncommon for an instrument to respond to these messages.

A polyphonic-key pressure message consists of 3 bytes of information, Figure 16.7: Polyphonic-key pressure status/MIDI channel number, MIDI note number, and pressure value.

Program-Change Messages. Program-change messages are used to change a MIDI instrument or device's active program or preset number. A preset is a user- or factory-defined number that actively

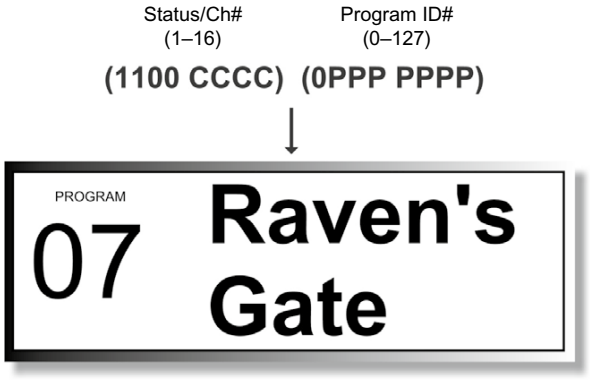


FIGURE 16.8
Byte structure of a MIDI program-change message.

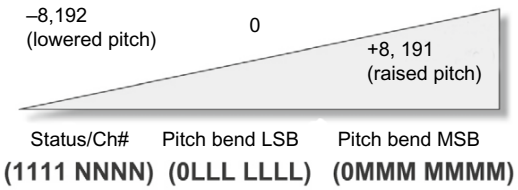


FIGURE 16.9
Byte structure of a pitch-bend message.

selects a specific sound patch or system setup. Using this extremely handy message, up to 128 presets can be remotely selected from another device or controller. For example:

- A program-change message can be transmitted from a remote keyboard or controller to an instrument, allowing sound patches to be remotely switched, Figures 16.8 and 16.9.
- Program-change messages could be programmed at the beginning of a sequence, so as to instruct the various instruments or voice generators to set to the correct sound patch before playing.
- It could be used to alter patches on an effects device, either in the studio or on stage. The list goes on.

A program-change message, Figure 16.8, consists of 2 bytes of information: program-change status/MIDI channel number and program ID number.

Pitch-bend Messages. Pitch-bend sensitivity refers to the response sensitivity (in semitones) of a pitch-bend wheel or other pitch-bend controlle, which, as you'd expect, is used to bend the pitch of a note upward or downward. Since the ear can be extremely sensitive to changes in pitch, this control parameter is encoded using 2 data bytes, yielding a total of 16,384 steps. Since this parameter is most commonly affected by varying a pitch wheel, Figure 16.9, the control values range from −8,192 to +8,191, with 0 being the instrument's or part's unaltered pitch.

Control-Change Messages. Control-change messages are used to transmit information to a device (either internally or through a MIDI line/network) that relates to real-time control over its performance parameters.

Three types of control-change messages can be transmitted via MIDI:

1. Continuous controllers: Controllers that relay a full range of variable control settings (often ranging in value between 0–127 although, in certain cases, two controller messages can be combined in tandem to achieve a greater resolution).
2. Switch controllers: Controllers that have either an off or an on state with no intermediate settings.

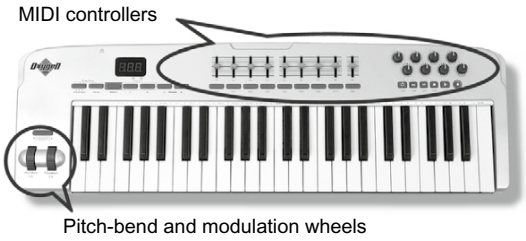


FIGURE 16.10
M-audio controller. (Courtesy of M-Audio, a division of Avid Technology, Inc., www.m-audio.com.)

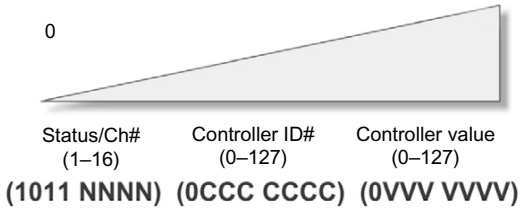


FIGURE 16.11
Byte structure of a control-change message.

3. Channel-mode message controllers: The final set of control change messages range between controller numbers 120 through 127, and are used to set the note sounding status, instrument reset, local control on/off, all notes off, and MIDI mode status of a device or instrument.

A single control-change message or a stream of such messages is transmitted whenever controllers (such as foot switches, foot pedals, pitch-bend wheels, modulation wheels, breath controllers, etc.) are varied in real-time. Newer controllers and software editors often offer up a wide range of switched and variable controllers, allowing for extensive, user-programmable control over any number of device, voice, and mixing parameters in real-time, Figure 16.10.

A control-change message, Figure 16.11, consists of 3 bytes of information: control-change status/MIDI channel number, controller ID number, and corresponding controller value.

As you can see, the second byte of the control-change message is used to denote the controller ID number. This all-important value is used to specify which of the device's program or performance parameters are to be addressed.

Table 16.2 details the general categories and conventions for assigning controller numbers to an associated parameter, as specified by the 1995 update of the MMA (MIDI Manufacturers Association, www.midi.org). This is definitely an important section to earmark, as these numbers will be an important guide towards knowing and/or finding the right ID number that can help you on your path towards finding that perfect variable for making it sound right.

The third byte of the control-change message is used to denote the controller's actual data value. This value is used to specify the position, depth, or level of a parameter. Here are a few examples as to how these values can be implemented to vary control and mix parameters.

In certain cases, greater resolutions than can be given by a single 7-bit course message (128 steps) might be available to increase a controller's resolution. This is simply accomplished by adding an additional fine controller value message to the data stream, resulting in an overall resolution that yields an overall total of 16,384 discrete steps!

System messages

System Messages. As the name implies, system messages are globally transmitted to every MIDI device in the MIDI chain. This is accomplished because MIDI channel numbers aren't addressed within the byte structure of a system message. Thus, any device will respond to these messages, regardless of its

Table 16.2 **Listing of Controller ID Numbers, Outlining Both the Defined Format and Convention and Controller Assignments.**

Control Number	Parameter
14 Bit Controllers Coarse/MSB (most significant bit)	
0	Bank Select 0–127 MSB
1	Modulation Wheel or Lever 0–127 MSB
2	Breath Controller 0–127 MSB
3	Undefined 0–127 MSB
4	Foot Controller 0–127 MSB
5	Portamento Time 0–127 MSB
6	Data Entry MSB 0–127 MSB
7	Channel Volume (formerly Main Volume) 0–127 MSB
8	Balance 0–127 MSB
9	Undefined 0–127 MSB
10	Pan 0–127 MSB
11	Expression Controller 0–127 MSB
12	Effect Control 1 0–127 MSB
13	Effect Control 2 0–127 MSB
14	Undefined 0–127 MSB
15	Undefined 0–127 MSB
16–19	General Purpose Controllers 1–4 0–127 MSB
20–31	Undefined 0–127 MSB
14-bit Controllers Fine/LSB (least significant bit)	
32	LSB for Control 0 (Bank Select) 0–127 LSB
33	LSB for Control 1 (Modulation Wheel or Lever) 0–127 LSB
34	LSB for Control 2 (Breath Controller) 0–127 LSB
35	LSB for Control 3 (Undefined) 0–127 LSB
36	LSB for Control 4 (Foot Controller) 0–127 LSB
37	LSB for Control 5 (Portamento Time) 0–127 LSB
38	LSB for Control 6 (Data Entry) 0–127 LSB
39	LSB for Control 7 (Channel Volume, formerly Main Volume) 0–127 LSB
40	LSB for Control 8 (Balance) 0–127 LSB
41	LSB for Control 9 (Undefined) 0–127 LSB
42	LSB for Control 10 (Pan) 0–127 LSB
43	LSB for Control 11 (Expression Controller) 0–127 LSB
44	LSB for Control 12 (Effect control 1) 0–127 LSB
45	LSB for Control 13 (Effect control 2) 0–127 LSB
46–47	LSB for Control 14–15 (Undefined) 0–127 LSB
48–51	LSB for Control 16–19 (General Purpose Controllers 1–4) 0–127 LSB
52–63	LSB for Control 20–31 (Undefined) 0–127 LSB

7-bit Controllers	
64	Damper Pedal On/Off (Sustain) <63 off, >64 on
65	Portamento On/Off <63 off, >64 on
66	Sustenuto On/Off <63 off, >64 on
67	Soft Pedal On/Off <63 off, >64 on
68	Legato Footswitch <63 Normal, >64 Legato
69	Hold 2 <63 off, >64 on
70	Sound Controller 1 (Default: Sound Variation) 0–127 LSB
71	Sound Controller 2 (Default: Timbre/Harmonic Intens.) 0–127 LSB
72	Sound Controller 3 (Default: Release Time) 0–127 LSB
73	Sound Controller 4 (Default: Attack Time) 0–127 LSB
74	Sound Controller 5 (Default: Brightness) 0–127 LSB
75	Sound Controller 6 (Default: Decay Time—see MMA RP-021) 0–127 LSB
76	Sound Controller 7 (Default: Vibrato Rate—see MMA RP-021) 0–127 LSB
77	Sound Controller 8 (Default: Vibrato Depth—see MMA RP-021) 0–127 LSB
78	Sound Controller 9 (Default: Vibrato Delay—see MMA RP-021) 0–127 LSB
79	Sound Controller 10 (Default undefined—see MMA RP-021) 0–127 LSB
80–83	General Purpose Controller 5–8 0–127 LSB
84	Portamento Control 0–127 LSB
85–90	Undefined
91	Effects 1 Depth (Default: Reverb Send Level) 0–127 LSB
92	Effects 2 Depth (Default: tremolo Level) 0–127 LSB
93	Effects 3 Depth (Default: Chorus Send Level) 0–127 LSB
94	Effects 4 Depth (Default: Celeste [Detune] Depth) 0–127 LSB
95	Effects 5 Depth (Default: Phaser Depth) 0–127 LSB
Parameter Value Controllers	
96	Data Increment (Data Entry +1)
97	Data Decrement (Data Entry -1)
98	Non-Registered Parameter Number (NRPN)—LSB 0–127 LSB
99	Non-Registered Parameter Number (NRPN)—MSB 0–127 MSB
100	Registered Parameter Number (RPN)–LSB* 0–127 LSB
101	Registered Parameter Number (RPN)—MSB* 0–127 MSB
102–119	Undefined
Reserved for Channel Mode Messages	
120	All Sound Off 0
121	Reset All Controllers
122	Local Control On/Off 0 off, 127 on
123	All Notes Off
124	Omni Mode Off (+ all notes off)
125	Omni Mode On (+ all notes off)
126	Poly Mode On/Off (+ all notes off)
127	Poly Mode On (+ mono off +all notes off)

MIDI channel assignment. The three system message types are system-common messages, system real-time messages, and system-exclusive messages.

System-Common Messages. System-common messages are used to transmit MIDI time code, song position pointer, song select, tune request, and end-of-exclusive data messages throughout the MIDI system or 16 channels of a specified MIDI port.

MTC Quarter-Frame Messages. MIDI time code (MTC) provides a cost-effective and easily implemented way to translate SMPTE (a standardized synchronization time code) into an equivalent code that conforms to the MIDI 1.0 spec. It allows time-based codes and commands to be distributed throughout the MIDI chain in a cheap, stable, and easy-to-implement way. MTC quarter-frame messages are transmitted and recognized by MIDI devices that can understand and execute MTC commands.

A grouping of eight quarter frames is used to denote a complete time code address (in hours, minutes, seconds, and frames), allowing the SMPTE address to be updated every two frames. Each quarter-frame message contains 2 bytes. The first is a quarter-frame common header, while the second byte contains a 4-bit nibble that represents the message number (0–7). A final nibble is used to encode the time field (in hours, minutes, seconds, or frames).

Song Position Pointer Messages. As with MIDI time code, song position pointer (SPP) lets you synchronize a sequencer, tape recorder, or drum machine to an external source from any measure position within a song. The SPP message is used to reference a location point in a MIDI sequence (in measures) to a matching location within an external device. This message provides a timing reference that increments once for every six MIDI clock messages (with respect to the beginning of a composition).

Unlike MTC (which provides the system with a universal address location point), SPP's timing reference can change with tempo variations, often requiring that a special tempo map be calculated in order to maintain synchronization. Because of this fact, SPP is used far less often than MIDI time code.

Song Select Messages. Song select messages are used to request a specific song from a drum machine or sequencer (as identified by its song ID number). Once selected, the song will thereafter respond to MIDI start, stop, and continue messages.

Tune Request Messages. The tune request message is used to request that a MIDI instrument initiate its internal tuning routine (if so equipped).

End-of-Exclusive Messages. The transmission of an end-of-exclusive (EOX) message is used to indicate the end of a system-exclusive message. In-depth coverage of system-exclusive messages will be discussed later in this chapter.

System Real-Time Messages. Single-byte system real-time messages provide the all-important timing element required to synchronize all of the MIDI devices in a connected system. To avoid timing delays, the MIDI specification allows system real-time messages to be inserted at any point in the data stream, even between other MIDI messages.

Timing-Clock Messages. The MIDI timing-clock message is transmitted within the MIDI data stream at various resolution rates. It is used to synchronize the internal timing clocks of each MIDI device within the system and is transmitted in both the start and stop modes at the currently defined tempo rate.

In the early days of MIDI, these rates (which are measured in pulses per quarter note, ppq) ranged from 24 to 128 ppq. However, continued advances in technology have brought these rates up to 240, 480, or even 960 ppq.

Start Messages. Upon receipt of a timing-clock message, the MIDI start command instructs all connected MIDI devices to begin playing from their internal sequences initial start point. Should a program be in midsequence, the start command will reposition the sequence back to its beginning, at which point it will begin to play.

Stop Messages. Upon receipt of a MIDI stop command, all devices within the system will stop playing at their current position point.

Continue Messages. After receiving a MIDI stop command, a MIDI continue message will instruct all connected devices to resume playing their internal sequences from the precise point at which they were stopped.

Active-Sensing Messages. When in the stop mode, an optional active-sensing message can be transmitted throughout the MIDI data stream every 300 milliseconds. This instructs devices that can recognize this message that they're still connected to an active MIDI data stream.

System-Reset Messages. A system-reset message is manually transmitted in order to reset a MIDI device or instrument back to its initial power-up default settings (commonly mode 1, local control on, and all notes off).

System-Exclusive Messages. The system-exclusive (SysEx) message allows MIDI manufacturers, programmers and designers to communicate customized MIDI messages between MIDI devices. It's the purpose of these messages to give manufacturers, programmers, and designers the freedom to communicate any device-specific data of an unrestricted length, as they see fit. In practice, SysEx data is commonly used to communicate real-time controller information (i.e., a remote controller surface will commonly use SysEx to communicate data to/from a MIDI-capable hard- or software device. SysEx can also be used transmit and receive device-specific program, patch parameter and sample data from one instrument or device to another. For example, SysEx can be used to transmit patch and overall setup data between identical make and (most-often) model of synthesizer. Let's say that you have a Brand X Model Z synthesizer and it turns out that you have a buddy across town who also has a Brand X Model Z. That's cool, except your buddy's synth has a completely different set of sound patches that was loaded into their instrument and you want them! SysEx to the rescue! All you need to do is go over and transfer your buddy's patch data into your synth, or into a MIDI sequencer as a SysEx data dump. In order to make life easier, make sure you take your instruction manual along (just in case you run into a snag), and follow these simple guidelines. I'll caution you that you're taking on these tasks at your own risk. Take your time; be patient and be careful during these procedures:

1. Back up your present patch data! This can be done by transmitting a SysEx dump of your synthesizer's entire patch and setup data to your sequencer's SysEx dump utility, or SysEx track on your sequencer (of course, you should get out both the device's manual and your sequencer's manual and follow their SysEx dump instructions very carefully during the process). This is so important that I'll say it again: Back up your present patch data before attempting a SysEx dump! If you forget and download a new SysEx dump, your previous settings could easily be lost.
2. Save the data, according to your sequencer's manual.
3. Check that the dump was successful by reloading it back into the device in question. Did it reload properly? If so, your current patch data is now saved.
4. Next, connect your buddy's device to your sequencer. Dump this data to your sequencer. Save the new patch data (using a new and easily identifiable file name), according to your sequencer's manual and then safely back this data up.
5. Reconnect the sequencer to your synth and load the new data dump into it. Does your synth have a bunch of new sounds? Now reload your original SysEx dump back into your device. Are the original sounds restored?

The transmission format of a SysEx message, Figure 16.12, as defined by the MIDI standard, includes a SysEx status header, manufacturer's ID number, any number of SysEx data bytes, and an EOX byte. On receiving a SysEx message, the identification number is read by a MIDI device to determine whether or not the following messages are relevant. This is easily accomplished, because a unique 1- or 3-byte ID number is assigned to each registered MIDI manufacturer. If this number doesn't match the receiving

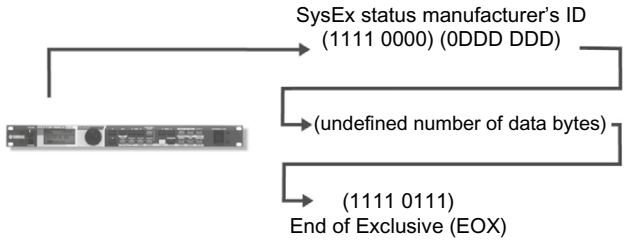


FIGURE 16.12
System-exclusive data (one ID byte format).

MIDI device, the ensuing data bytes will be ignored. Once a valid stream of SysEx data is transmitted, a final EOX message is sent, after which the device will again begin responding to incoming MIDI performance messages.

HARDWARE SYSTEMS WITHIN MIDI PRODUCTION

As a data transmission medium, MIDI is relatively unique in the world of sound production in that it's able to pack 16 discrete channels of performance, controller, and timing information and transmit it in one direction, using data densities that are economically small and easy to manage. In this way, it's possible for MIDI messages to be communicated from a specific source (such as a keyboard or MIDI sequencer) to any number of devices within a connected network over a single MIDI data chain. In addition, MIDI is flexible enough that multiple MIDI data lines can be used to interconnect devices in a wide range of possible system configurations (for example, multiple MIDI lines can be used to transmit data to instruments and devices over 32, 48, 128, or more discrete MIDI channels!)

The MIDI Cable. A MIDI cable, Figure 16.13, consists of a shielded twisted pair of conductor wires that has a male 5-pin DIN plug located at each of its ends. The MIDI specification currently uses only three of the five pins, with pins 4 and 5 being used as conductors for MIDI data, while pin 2 is used to connect the cable's shield to equipment ground. Pins 1 and 3 are currently not in use, although the next section describes an ingenious system for power devices through these pins, using a system that's known as MIDI phantom power. The cables themselves use twisted cable and metal shield groundings to reduce outside interference, such as radio-frequency interference (RFI) or electrostatic interference, both of which can serve to distort or disrupt the transmission of MIDI messages.

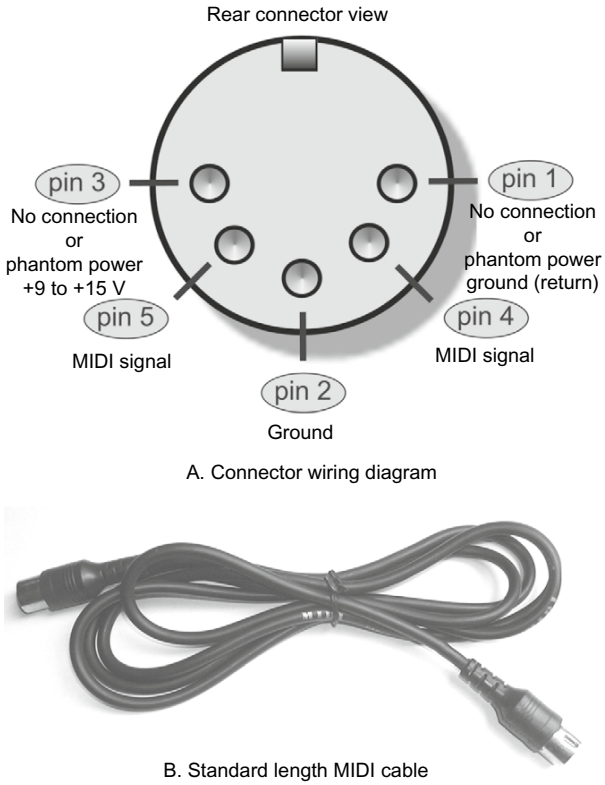


FIGURE 16.13
The MIDI cable.

MIDI cables come prefabricated in lengths of 2, 6, 10, 20, and 50 feet, and can commonly be obtained from music stores that specialize in MIDI equipment. To reduce signal degradations and external interference that tends to occur over extended cable runs, 50 feet is the maximum length specified by the MIDI specification. (As an insider tip, I found that Radio Shack is also a great source for picking up 3 and 6 feet MIDI cables at a fraction of what you'd sometimes spend at a music store).

MIDI Phantom Power. In December 1989, Craig Anderton wrote an article in *Electronic Musician* about a proposed idea for allowing a source to provide a standardized 12 Vdc power supply to instruments and MIDI devices directly through pins 1 and 3 of a basic MIDI cable. Although pins 1 and 3 are technically reserved for possible changes in future MIDI applications, over the years several forward-thinking manufacturers (and project enthusiasts) have begun to implement MIDI phantom power directly into their studio and on-stage systems.

Wireless MIDI. In recent times, a number of companies have begun to manufacturer wireless MIDI transmitters that can allow a battery-operated MIDI guitar, wind controller, etc. to be footloose and fancy free on-stage and in the studio. Working at distances of up to 500 feet, these battery-powered transmitter/receiver systems introduce very low delay latencies and can be switched over a number of radio channel frequencies.

MIDI Jacks. MIDI is distributed from device to device using three types of MIDI jacks: MIDI In, MIDI Out, and MIDI Thru, Figure 16.14. These three connectors use 5-pin DIN jacks as a way to connect MIDI instruments, devices, and computers into a music and/or production network system. As a side note, it's nice to know that these ports (as strictly defined by MIDI 1.0 Spec.) are optically isolated to eliminate possible ground loops that might occur when connecting numerous devices together.

- **MIDI In**—The MIDI In jack receives messages from an external source and communicates this performance, control, and/or timing data to the device's internal microprocessor, allowing an instrument to be played and/or a device to be controlled. More than one MIDI In jack can be designed into a system to provide for MIDI merging functions or for devices that can support more than 16 channels (such as a MIDI Interface). Other devices (such as a controller) might not have a MIDI In jack at all.
- **MIDI Out**—The MIDI Out jack is used to transmit MIDI performance, control messages or SysEx from one device to another MIDI instrument or device. More than one MIDI Out jack can be designed into a system, giving it the advantage of controlling and distributing data over multiple MIDI paths using more than just 16 channels (i.e., 16 channels $\times$ *N* MIDI port paths).
- **MIDI Thru**—The MIDI Thru jack retransmits an exact copy of the data that's being received at the MIDI In jack. This process is important, because it allows data to pass directly through an instrument or device to the next device in the MIDI chain. Keep in mind that this jack is used to relay an exact copy of the MIDI In data stream and isn't merged with data being transmitted from the MIDI Out jack.
- **MIDI Echo**—Certain MIDI devices may not include a MIDI Thru jack, at all. Certain of these devices, however, may give the option of switching the MIDI Out between being an actual MIDI Out jack and a MIDI Echo jack, Figure 16.15. As with the MIDI Thru jack, a MIDI echo option can be used to retransmit an exact copy of any information that's received at the MIDI In port and route this data to the MIDI Out/Echo jack. Unlike a dedicated MIDI Out jack, the MIDI

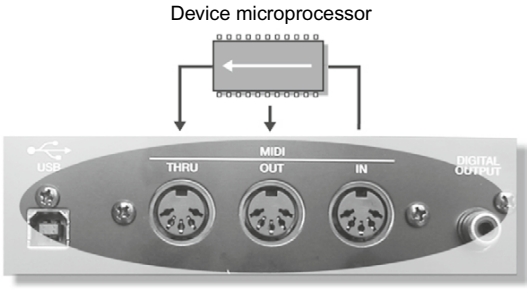


FIGURE 16.14
MIDI in, out, and thru ports, showing the device's signal path routing.

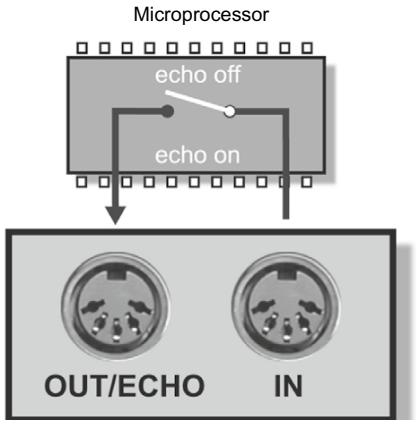


FIGURE 16.15
MIDI echo configuration.

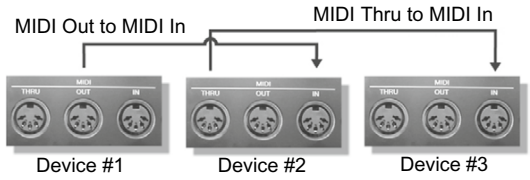


FIGURE 16.16
The two valid means of connecting one MIDI device to another.

Echo function can often be selected to merge incoming data with performance data that's being generated by the device itself. In this way, more than one controller can be placed in a MIDI system at one time. It should be noted that although performance and timing data can be echoed to a MIDI Out/Echo jack, not all devices can echo SysEx data.

Typical Configurations. Although electronic studio production equipment and setups are rarely alike (or even similar), there are a number of general rules that make it easy for MIDI devices to be connected into a functional network. These common configurations allow MIDI data to be distributed in the most efficient and understandable manner possible.

As a primary rule, there are only two valid ways to connect one MIDI device to another within a MIDI chain, Figure 16.16:

1. Connecting the MIDI Out jack of a source device (controller or sequencer/computer) to the MIDI In of a second device in the chain.
2. Connecting the MIDI Thru jack of the second device to the MIDI In jack of the third device in the chain and following this same Thru-to-In convention until the end of the chain is reached.

The Daisy Chain. One of the simplest and most common ways to distribute data throughout a MIDI system is the daisy chain. This method relays MIDI data from a source device (controller or sequencer/computer) to the MIDI In jack of the next device in the chain (which receives and acts upon this data). In turn, this device relays an exact copy of this incoming data out to its MIDI Thru jack, which is then relayed to the next device in the chain. This device can then relay an exact copy of this incoming data out to its MIDI Thru jack, which is then relayed to the next device in the chain… etc. In this way, up to 16 channels of MIDI data can be chained from one device to the next within a connected data network—and it's precisely this concept of transmitting multiple channels through a single MIDI line that makes this concept work! Let's try to understand this concept better by looking at a few examples.

Figure 16.17A shows a simple (and common) example of a MIDI daisy chain, whereby data flows from a controller (MIDI Out jack of the source device) to a synth module (MIDI In jack of the second device in the chain), where an exact copy of this data is relayed from its MIDI Thru jack to another synth

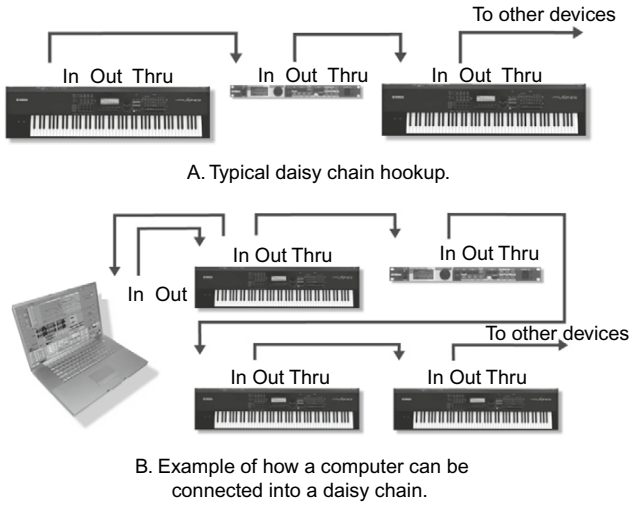


FIGURE 16.17
Example of a connected MIDI system using a daisy chain.

(MIDI In jack of the third device in the chain). It shouldn't be hard to understand that if our controller is transmitting on MIDI channel 2, the second synth in the chain (which is set to channel 2) will ignore the messages and not play while the 3rd synth (which is set to channel 3) will be playing its heart out. The moral of this story is that although there's only one connected data line, a wide range of instruments and channel voices can be played in a surprisingly large number of combinations, all by using individual channel assignments along a daisy chain.

Another example, Figure 16.17b, shows how a computer can easily be designated as the master source within a daisy chain, so that a sequencing program could be used to control the entire playback and channel routing functions of a daisy-chained system. In this situation, the MIDI data flows from a master controller/synth to the MIDI In jack of a computer's MIDI interface—where the data can be played into, processed, and rechannelized through a MIDI sequencer. The MIDI Out of the interface is then routed back to the MIDI In jack of the master controller/synth (which receives and acts on this data). In turn, the controller relays an exact copy of this incoming data out to its MIDI Thru jack, which is then relayed to the next device in the chain. This device can then relay an exact copy of this incoming data out to its MIDI Thru jack, which is then relayed to the next device in the chain, etc. When we stop to think about this second example, the controller is used to perform into the MIDI sequencer, which then is used to communicate this edited and processed performance data out to the various instruments throughout the connected MIDI chain.

The Multiport Network. Another common approach to routing MIDI throughout a production system involves distributing MIDI data through the multiple 2, 4 and 8 In/Out ports that are available on a newer multiport MIDI interfaces or through the use of multiple MIDI interfaces (typically these are USB devices).

In larger, more complex, MIDI systems, a multiport MIDI network, Figure 16.17, offers several advantages over a single daisy chain path. One of the most important is its ability to address devices within a complex setup that requires more than 16 MIDI channels. For example, a 2 × 2 MIDI interface that offers up two-independent In/Out paths is capable of addressing up to 32 channels simultaneously (i.e., port A 1–16 and port B 1–16), whereas an 8 × 8 port interface is capable of addressing up to 128 individual MIDI channels.

The MIDI interface

Although computers and electronic instruments both communicate using the digital language of 1s and 0s, computers simply can't understand the language of MIDI without the use of a device that translates the serial messages into a data structure that computers can comprehend. Such a device is known as the *MIDI interface.*

FIGURE 16.18
M-Audio MIDISPORT 4 × 4 MIDI interface. (Courtesy of M-Audio, a division of Avid Technology, Inc., www.m-audio.com.)

A wide range of MIDI interfaces currently exist that can be used with most computer systems and OS platforms. For the casual and professional musician, interfacing MIDI into a production system can be done in a number of ways. Probably the most common way to access MIDI In, Out, and Thru jacks is on a modern-day USB or FireWire audio interface or instrument/DAW controller surface. It's become a common matter for portable devices to offer 16 channels of I/O (on one port), while multi-channel interfaces often include multiple MIDI I/O ports that can give you access to 32 or more channels.

Another additional option is to choose a USB MIDI interface that can range from devices that include a single I/O port (16 channels) to a multiport system that can easily handle up to 128 channels over eight I/O ports. The multiport MIDI interface, Figure 16.18, is often the device of choice for most professional electronic musicians who require added routing and synchronization capabilities. These rack-mountable USB devices can be used to provide eight independent MIDI Ins and Outs to easily distribute MIDI and time code data through separate lines over a connected network.

Hardware and software electronic instruments

Since its inception in the early 80s, MIDI-based electronic musical instruments have helped to shape the face and sounds of our modern music culture. These devices (along with digital audio and advances in recording equipment technology) have altered music production, through the creation of one of the most cost-effective and powerful tools in the development of music history—the personal project studio.

The following is a sample listing of the many hardware MIDI instrument types that are currently available on the market.

The Synth. A synthesizer, Figure 16.19, is an electronic instrument that uses multiple sound generators to create complex waveforms that can be combined (using various waveform synthesis techniques) into countless sonic variations. These synthesized sounds have become a basic staple of modern music and vary from sounding cheesy, to those that closely mimic traditional instruments all the way, to those that generate rich, otherworldly sounds that literally defy classification.

Synthesizers (also known as synths) generate sounds and percussion sets using a number of different technologies or program algorithms. The earliest synths were analog in nature and generated sounds using additive or subtractive FM (frequency modulation) synthesis. This process generally involves the use of at least two signal generators (commonly referred to as *operators*) to create and modify a voice. Often, this is done through the analog or digital generation of a signal that modulates or changes the tonal and amplitude characteristics of a base carrier signal. More sophisticated FM synths can use up to

FIGURE 16.19
Bass Station analog bass synth. (Courtesy of Novation Digital Music Systems, Ltd.; www.novationmusic.com.)

FIGURE 16.20
Yamaha MOTIF-RACK ES synth. (Courtesy of Yamaha Corporation of America, www.yamaha.com.)

four or six operators per voice and also often use filters and variable amplifier types to alter the signal's characteristics into a sonic voice that either roughly imitates acoustic instruments or creates sounds that are totally unique.

Another technique that's used to create sounds is wavetable synthesis. This technique works by storing small segments of digitally sampled sound into a read-only memory chip. Various sample-based synthesis techniques use sample looping, mathematical interpolation, pitch shifting, and digital filtering to create extended and richly textured sounds that use a very small amount of sample memory.

Synthesizers are also commonly designed into rack- or half-rack-mountable modules, Figure 16.20, that contain all of the features of a standard synthesizer, except that they don't incorporate a keyboard controller. This space-saving feature means that more synths can be placed into your system and can be controlled from a master keyboard controller or sequencer, without cluttering up the studio with redundant keyboards.

Software Synthesis and Sample Re-synthesis. Since wavetable synthesizers derive their sounds from prerecorded samples that are stored in a digital memory medium, it logically follows that these sounds can also be stored on hard disk (or any other medium) and loaded into the RAM memory of a personal computer. This process of downloading wavetable samples into a computer and then manipulating these samples is used to create what is known as a virtual or software synthesizer, Figure 16.21.

In recent years, software synths have grown from being novel and obscure programs that were primarily used by the academic community to their present state of being widely accepted in the production community as a cost-effective musical instrument. These software modules can be used in conjunction with a digital audio workstation to offer up a wide range of complex sounds that can mimic traditional instruments, as well as create sonic textures that are both new and interesting.

Sample re-synthesis software systems are able to take software synthesis to a new level, by allowing the user to build, save, and recall sonic patches that can be built from traditional synthesis building blocks (such as oscillators, voltage-controlled amplifiers, voltage-controlled filters, and mixers). In addition to sound generation, digital audio samples can be imported and re-synthesized in a way that can create sounds of almost any texture or type that you can possibly imagine. All of these software blocks can be combined in a graphic environment that allows these instruments, textures, and soundscapes to be easily saved to disk for later recall.

Using various internal software data communications protocols, it's possible to communicate MIDI, audio, timing sync and control data between an instrument (or effect plug-in) and a host DAW program/

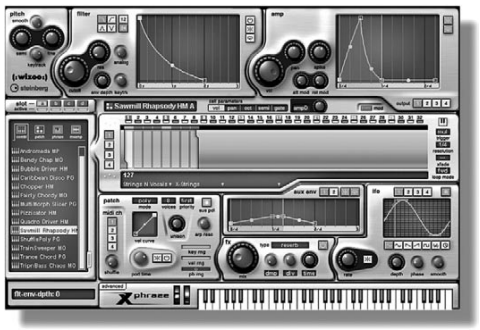


FIGURE 16.21
Steinberg xphrase VSTi software synth. (Courtesy of Steinberg Media Technologies GmbH, a division of Yamaha Corporation, www.steinberg.net.)

FIGURE 16.22
Akai MPC-1000 Music Production Center. (Courtesy of Akai Professional, www.akaipro.com.)

CPU processor. These plug-in protocols make it possible for much or all of the audio and timing data to be routed through the host audio application, allowing the instrument or application to either integrate into the DAW or application or to work in tandem so as to route the audio and performance/control data through the host application with relative ease. A few of these protocols include:

- Steinberg's VST (Virtual Studio Technology)
- MOTU's MAS (MOTU Audio System)
- Propellerheads ReWire.

Samplers. A sampler, Figure 16.22, is a device that can convert audio into a digital form and/or manipulate prerecorded sampled data, using the system's own random access memory (RAM). Once loaded into RAM, the sampled audio can be edited, transposed, processed, and played in a polyphonic musical fashion.

Basically, a sampler can be thought of as a wavetable synth that lets you record, load, and edit samples into RAM memory. Once loaded, these sounds (whose length and complexity are often limited only by memory size and your imagination) can be looped, modulated, filtered, and amplified (according to user or factory setup parameters), in a way that allows the waveshapes and envelopes to be modified. Signal processing capabilities, such as basic editing, looping, gain changing, reverse, sample-rate conversion, pitch change, and digital mixing capabilities can also be altered and/or varied.

A hardware sampler's design will often include a keyboard or set of trigger pads that let you polyphonically play samples as musical chords, sustain pads, triggered percussion sounds, or sound effect events. These samples can be played according to the standard Western musical scale (or any other scale, for that matter) by altering the playback sample rate over the controller's note range. For example, pressing a low-pitched key on the keyboard will cause the sample to be played back at a lower sample rate, while pressing a high-pitched one will cause the sample to be played back at rates that would put Mickey Mouse to shame. By choosing the proper sample rate ratios, sounds can be polyphonically played (whereby multiple notes are sounded at once) at pitches that correspond to standard musical chords and intervals.

A sampler (or synth) with a specific number of voices (i.e., 64 voices) simply means that up to 64 notes can be simultaneously played on a keyboard at any one time. Each sample in a multiple-voice system can be assigned across a performance keyboard, using a process known as *splitting* or *mapping*. In this way, a sound can be assigned to play across the performance surface of a controller over a range of notes, known as a zone, Figure 16.23. In addition to grouping samples into various zones, velocity can enter into the equation by allowing multiple samples to be layered across the same keys of a controller, according to how soft or hard they are played. For example, a single key might be layered so that pressing the key lightly would reproduce a softly recorded sample, while pressing it harder would produce a louder sample with a sharp percussive attack. In this way, mapping can be used to create a more realistic instrument or wild set of soundscapes that change not only with the played keys, but with velocity ranges as well.

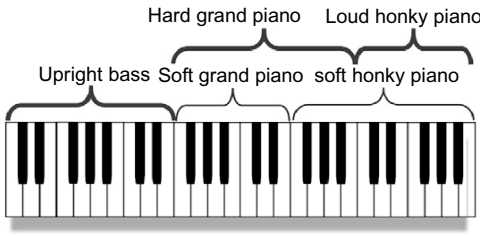


FIGURE 16.23
Samples can be mapped to various zones on a keyboard.

FIGURE 16.24
Steinberg's HALion VST software sampler. (Courtesy of Steinberg Media Technologies GmbH, a division of Yamaha Corporation, www.steinberg.net.)

In addition to hardware sampling systems, a growing number of virtual or software samplers exist that use a computer's existing memory, processing, and signal routing capabilities in order to polyphonically reproduce samples in real time.

Offering much of the same functionality as their hardware counterparts, these software-based systems, Figure 16.24, are capable of editing, mapping, and splitting sounds across a MIDI keyboard, using on-screen graphic controls and DAW integration that has improved to the point of equaling or surpassing their hardware counterparts in cost-effectiveness, power, and ease of use.

As with a software synth, software samplers derive their sounds from recorded and/or imported audio data that is stored as digital audio data within a personal computer. Using the DSP capabilities of today's computers (as well as the recording, sequencing, processing, mixing, and signal routing capabilities of most digital audio workstations), most software samplers are able to store and access samples within the internal memory of a laptop or desktop computer. Using a graphic interface, these sampling systems often allow the user to:

- Import previously recorded soundfiles (often in WAV, AIF, and other common formats)
- Edit and loop sounds into a usable form
- Vary envelope parameters (i.e., dynamics over time)
- Vary processing parameters
- Save the edited sample performance setup as a file for later recall.

Software sampler systems are also often able to communicate MIDI, audio, timing sync and control data between a hard- or software instrument and a host DAW program/CPU processor, allowing for a wide range of control and setup recall.

The Drum Machine. The drum machine is most commonly a sample-based digital audio device that can't record audio into its internal memory (although this has changed in recent years, allowing it to import, record, and manipulate sampled audio much like a sampler). Traditionally, these hardware or software systems use ROM-based, prerecorded samples to reproduce high-quality drum sounds.

FIGURE 16.25
Groove Agent 3 VST Virtual Drummer. (Courtesy of Steinberg Media Technologies GmbH, a division of Yamaha Corporation, www.steinberg.net).

These factory-loaded sounds often include a wide assortment of drum sets, percussion sets, rare and wacky percussion hits, and effected drum sounds (i.e., reverberated, gated, etc.). Who knows, you might even encounter "Hit me!" screams from the venerable King of Soul—James Brown.

Most hardware drum machines allow prerecorded samples to be assigned to a series of playable keypads that are often located on the machine's top face. This provides a straightforward controller surface that usually includes velocity and aftertouch dynamics. Drum voices can be assigned to each pad and edited using such control parameters as tuning, level, output assignment, and panning position. Multiple outputs are often provided, enabling individual or groups of voices to be routed to a specific output on a mixer or console.

Although a number of hardware drum machine designs include a built-in sequencer, it's more likely that these workhorses will be triggered from a MIDI sequencer. This lets us take full advantage of the real-time performance and editing capabilities that a sequencer has to offer. For example, sequenced patterns can easily be created in step time (where notes are entered and assembled into a rhythmic pattern one note at a time) and can then link together into a song that's composed of several rhythmic patterns. Alternately, performing into a sequencer on-the-fly can help create a live feel or you can combine step- and real-time tracks to create a human-sounding composite rhythm track. In the final analysis, the style and approach to composition is entirely up to you.

In addition to their hardware counterparts, an increasing number of software drum and groove instrument plug-ins have come onto the market that allow for drum patterns to be added to a production in a wide range of pattern and playing styles, Figure 16.25.

Performance and parameter controllers

MIDI performance controllers are used to translate the voicings and expressiveness of a musical performance into MIDI data, while a parameter controller surface is used to alter the control variables of a workstation, device or instrument.

It should be noted that a MIDI controller is expressly designed to control other devices (be they for sound, light or mechanical control) within a connected system. It contains no internal tone generators or sound-producing elements. Instead, it offers a wide range of controls for handling control, trigger and device switching events. In short, controllers have become an integral part of music production, and are available in many incarnations to control and emulate many types of musical instruments.

Keyboard Controller. The MIDI keyboard controller, Figure 16.26, is a keyboard device that's expressly designed to control hard/software synths, samplers, modules and other devices within a connected MIDI system. It contains no internal tone generators or sound-producing elements. Instead, its design includes a performance keyboard and controls for handling MIDI performance, control, and device switching events.

Percussion Controllers. MIDI percussion controllers are used to translate the voicings and expressiveness of a percussion performance into MIDI data. These devices are great for capturing the feel of a live performance, while giving you the flexibility of recording and automating a performance within a DAW/sequencer environment. These controllers vary over a wide range from being a simple and cost-effective setup (i.e., using the pads on a drum machine, keys on a keyboard surface, or pads on an intro-level drum controller) to a full-blown drum kit that mimics its acoustic cousin, Figures 16.27 and 16.28.

FIGURE 16.26
Novation ReMOTE 25SL MIDI Controller/Keyboard. (Courtesy of Novation Digital Music Systems, Ltd, www.novationmusic.com.)

FIGURE 16.27
Trigger Finger 16-Pad MIDI Drum Control Surface. (Courtesy of M-Audio, a division of Avid Technology, Inc., www.m-audio.com.)

FIGURE 16.28
DM5 Electronic Drum Kit. (Courtesy of Alesis, www.alesis.com.)

Wind Controllers. MIDI wind controllers are expressly designed to bring the breath and key articulation of a woodwind or brass instrument to a MIDI performance. These controller types are used because many of the dynamic- and pitch-related expressions (such as breath and controlled pitch glide) simply can't be communicated from a standard music keyboard. In these situations, wind controllers can often help create a dynamic feel that's more in keeping with their acoustic counterparts by using an interface that provides special touch-sensitive keys, glide- and pitch-slider controls, and real-time breath sensors for controlling dynamics.

MIDI Guitars. Guitar players often work at stretching the vocabulary of their instruments beyond the traditional norm. They love doing nontraditional gymnastics using such tools of the trade as distortion, phasing, echo, feedback, etc. Due to advances in guitar pickup and microprocessor technology, it's also possible for the notes and minute inflections of guitar strings to be accurately translated into MIDI data. With this innovation, many of the capabilities that MIDI has to offer are available to the electric (and electronic) guitarist. For example, a guitar's natural sound can be layered with a synth pad that's been transposed down, giving it a rich thick sound that just might shake your boots. Alternately, recording a sequenced guitar track into a session would give a producer the option of changing and shaping the sound later in mixdown! On-stage program changes are also a big plus for the MIDI guitar, allowing the player to radically switch between guitar voices from the guitar or sequencer or by stomping on a MIDI foot controller.

SEQUENCERS

Apart from electronic musical instruments, one of the most important tools that can be found in the modern-day project studio is the MIDI sequencer. Basically, a sequencer is a digital device that's used to record, edit, reproduce, and distribute MIDI messages in a sequential fashion. Most sequencers function using a traditional track-based interface, separating different instruments, voices, beats, etc. in a way that makes it easier for us humans to view MIDI data as though they were linear tracks on a DAW or tape machine.

These virtual tracks contain MIDI-related performance and control events that are made up of such channel and system messages as note on/off, velocity, modulation, aftertouch, and program/continuous-controller messages. Once a performance has been recorded into a sequencer's memory, these events can be graphically (or audibly) edited into a musical performance, played back and saved to a digital storage media for recall at any time.

Integrated Sequencers. Some of the newer and more expensive keyboard synth and sampler designs include a built-in sequencer. These portable keyboard workstations have the advantage of letting you take both the instrument and sequencer on the road without having to drag a computer along.

Integrated sequencers are designed into an instrument for the sole purpose of sequencing MIDI data, and include integrated controls for performing sequence-specific functions. Ease of use and portability are often the advantages of a hardware sequencer, most of which are designed to emulate the basic functions of a tape transport (record, play, start/stop, fast forward, and rewind).

These devices generally offer a moderate amount of editing features, including note editing, velocity and other controller messages, program change, cut and paste and track merging capabilities, tempo changes, etc. Programming, track, and edit information is commonly viewed on a liquid crystal display (LCD) that's often limited in size and resolution and generally limits information to a single parameter or track at a time.

These sequencers often don't offer a wide range of editing tools beyond standard transport functions, punch-in/out commands and other basic edit tools. However, they're often more than adequate for capturing and reproducing a performance and can be integrated with other instruments that are connected in a MIDI chain.

Software Sequencers. By far, the most common sequencer type is the software MIDI sequencer. These programs or integrated components of a digital audio workstation take advantage of the versatility that a computer can offer in the way of speed, flexibility, digital signal processing, memory management, and signal routing.

Computer-based sequencers offer numerous functional advantages over their hardware counterparts. Among these are increased graphic capabilities (which often offers extensive control over track- and

transport-related functions), standard computer cut and paste techniques, an on-screen graphic environment (allowing easy manipulation of program and edit-related data), routing of MIDI to multiple ports in a connected system, and the graphic assignment of instrument voices via program change messages (not to mention the ability to save and recall files using standard computer memory media). Now, let's take a look at how these devices function.

A basic introduction to sequencers

When dealing with any type of sequencer, one of the most important concepts to grasp is the fact that these devices don't store sound directly—instead, they encode MIDI messages that instruct an instrument to play a particular note, over a certain channel, at a specific velocity and with any optional controller values. In other words, a sequencer stores music-related data commands that follow in a sequential order, which then tells instruments and/or devices how their voices are to be played and/or controlled. This simple (but important) fact means that the amount of encoded data is far less memory intensive than its hard disk audio or video recording counterparts and that the data overhead that's required by MIDI is very small. In short, a computer-based sequencer can simultaneously operate in a digital audio, digital video, processing environment without placing an additional significant load on a computer's CPU.

As you might expect, many sequencer types are currently on the market, with each offering its own set of advantages and disadvantages. It's also true that each sequencer has its own basic operating feel, and thus choosing the best tool and toy for the job or studio is totally up to you.

Recording. From a functional standpoint, a sequencer is used as a digital workspace for creating personal compositions in environments that range from the bedroom to more elaborate project studios. Whether they're hardware or software based, most sequencers use a working interface that's designed to emulate the traditional multitrack recording environment. A tapelike transport lets you move from one location to the next using standard Play, Stop, FF, REW and Rec command buttons. Beyond using traditional record-enable button(s) to arm selected recording track(s), all you need to do is select the MIDI input (source) and outputs (destination) ports, instrument/voice MIDI channel, instrument patch and other setup information, press the record button, and start playing.

Once you've finished laying down a track, you can jump back to any point in the sequence and listen to your original track while continuing to lay down additional MIDI tracks until the song begins to form.

Almost all sequencers are capable of punching in and out of record while playing a sequence. This common function lets you drop in and out of record on a track (or tracks) in real-time. Although punch-in/out points can often be manually performed on-the-fly, most sequencers can perform a punch automatically, once the in/out measure numbers have been graphically or numerically entered. The sequence can then be rolled back a few measures and the artist can play along, while the sequencer automatically performs the necessary switching functions (usually with multiple take and full undo capabilities).

In addition to recording a performance in a track-based environment, most sequencers let you enter note values into sequence one note at a time. This feature (known as step time) lets you give the sequencer a basic tempo and note length (i.e., quarter note, sixteenth note, etc.) and then manually enter the notes from a keyboard or other controller. This data entry style is often (but not always) used with fast, hi-tech and dance styles, where a real-time performance just isn't possible or accurate enough for the song.

Whether you're recording a track in real-time or in step-time, it's almost always best to select the proper song tempo before recording a sequence. I bring this up because most sequencers are able to output a click track that can be used as an accurate audible guide for keeping in time with the song's selected tempo. It's also critical that the tempo be accurate when trying to sync groove loops and rhythms to a sequence via plug-ins or external instruments.

Editing. One of the more important features that a sequencer (or sequenced MIDI track within a DAW) has to offer is its ability to edit tracks or blocks within a track. Of course, these editing functions and capabilities often vary between hardware and software sequencers.

The main track window of a sequencer or MIDI track on a DAW is used to display such track information as the existence of track data, track names, MIDI port assignments for each track, program change assignments, volume controller values, etc.

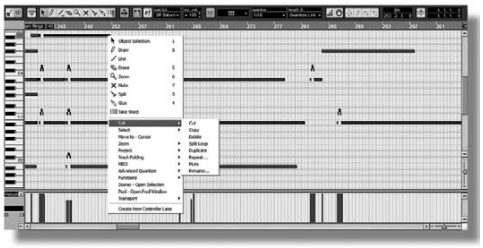


FIGURE 16.29
The presence of MIDI message data will often appear as a series of highlighted areas within a sequence track or a window. (Courtesy of Steinberg Media Technologies GmbH, a division of Yamaha Corporation, www.steinberg.net.)

Depending on the sequencer, the existence of MIDI data on a particular track at a particular measure point (or over a range of measures) is often indicated by the visual display of MIDI data in a piano-roll fashion (showing the general vertical and length placements of the notes as they progress though the musical passage) as shown in Figure 16.29.

By navigating around the various data display and parameter boxes, it's possible to use cut and paste and/or direct edit techniques to vary note, length and controller parameters for almost every facet of a section or musical composition. For example, let's say that we really screwed up a few notes when laying down an otherwise killer bass riff. With MIDI, fixing the problem is totally a no-brainer. Simply highlight each fudged note and drag it to it's proper note location. We can even change the beginning and end points in the process. In addition, tons of other parameters can be changed including velocity, modulation and pitch bend, note and song transposition, quantization, and humanizing (factors that eliminate or introduce human timing errors that are generally present in a live performance), as well as full control over program and continuous controller messages. The list goes on.

Playback. Once a composition is complete, all of the MIDI tracks in a project can be transmitted through the various MIDI ports and channels to plug-ins, instruments, or devices for playback. Since the data exists as encoded real-time control commands, you can listen to the sequence and make changes at any time. For example, you could change instrument settings (by changing or editing patch voices), alter volume and other mix changes, or experiment with such controllers as pitch bend, modulation or aftertouch, and even change the tempo and key signature. In short, this medium is infinitely flexible how a performance and/or set of parameters can be created, saved, folded, spindled, and mutilated until you've arrived at the sound and feel that you want.

Another of the greatest beauties of MIDI production is its ability to be altered at any later point in time. For example, let's say that 5 years ago you laid down a killer synth riff in a song that made it onto the charts. A couple of weeks ago a producer came to you in hopes of collaborating on a remix. Of course, technology marches on and your studio has improved over time. First off, even though a lot of the setup parameters have been saved with the original sequence, let's assume that you were smart enough to keep really good setup notes. One big change, however, is that you have a new software synth that has a patch that sounds better than the original patch. Since the remix is to be used in an upcoming film track, MIDI can be used to tweak things up a bit by splitting the riff into two parts: one that contains the lower notes and another the highs. By sending the lows to one patch on the synth and the highs to another, not only have you improved the overall sound, you've filled it out by expanding the soundfield into surround. Without MIDI, you'd have to arrange for a new session and hope that it all goes well; with MIDI, the performance is exactly the same and improvements are made in a no-brainer environment. This is what MIDI's all about—performance, repeatability, easy editing, and cost-effective power!

I now have to take time out to give you a few pointers that will make your life easier when dealing with MIDI production.

1. Remember to set the session to the proper tempo at the beginning of the session. Although tempo can be changed at a later time, attention to tempo details can help you to avoid later pitfalls.
2. Always name your track before you go into record (this goes for both audio and MIDI tracks). Properly naming your tracks (i.e., with its instrument, patch name) is the first step toward good documentation.

3. You can *never* overdocument a session. Keeping good instrument, patch, settings, musician, studio, and other notes might not only come in handy—it can save your butt if you need to revisit the tracks in the future.
4. Never delete a final take MIDI track from a DAW session. Even though you've transferred the instrument to an audio track, it is *always* wise to archive the original MIDI track with session. Trust me, both you and the producer will be glad you did, should any changes need to be made to the track in the future.

Other software sequencing applications

In addition to DAW and sequencing packages that are designed to handle most of the day-to-day production needs of the musician, other types of software tools and applications exist that can help to carry out specialized tasks. A few of these packages include drum pattern editors, algorithmic composition programs, patch editors and music printing programs.

Drum-Pattern Editor/Sequencers. At any one time, there are a handful of companies that have software or hardware devices that are specifically designed to create and edit drum patterns. In addition, most of the higher-end DAW audio production systems also include a drum pattern editor that relies on user input and quantization to construct and chain together any number of user-created percussion grooves. More often than not, these editors use a grid pattern that displays drum-related MIDI notes or subpatterns along the vertical axis, while time is represented in metric divisions along the horizontal axis, Figure 16.30. By clicking on each grid point with a mouse or other input system, individual drum or effect sounds can be built into rhythmic patterns.

Once created, these and other patterns can be linked together to create a partial or complete rhythm section within a song. These editors commonly offer such features as the ability to change MIDI note values (thereby changing drum voices), note length, quantization and humanization, as well as adjustments to note and pattern velocities. Once completed, the sequenced drum track (or chained patterns) can be imported into a sequence, saved, and/or exported.

Groove Tools. Getting into the groove of a piece of music often refers to a feeling that's derived from the underlying foundation of the piece: rhythm. With the introduction and maturation of MIDI and digital audio, new and wondrous tools have made their way into the mainstream of music production that can help us to use these technologies to forge, fold, mutilate and create compositions that make direct use of rhythm and other building blocks of music through the use of looping technology.

Of course, the cyclic nature of loops can be—repeat repeat—repetitive in nature, but new toys and techniques in looping have injected the notion of flexibility, real-time control, real-time processing, and mixing to new heights that can be used by an artist as a wondrously expressive tool.

Loop-based audio editors are groove-driven music programs, Figures 16.31 and 16.32, that are designed to let you drag and drop prerecorded or user-created loops and audio tracks into a graphic

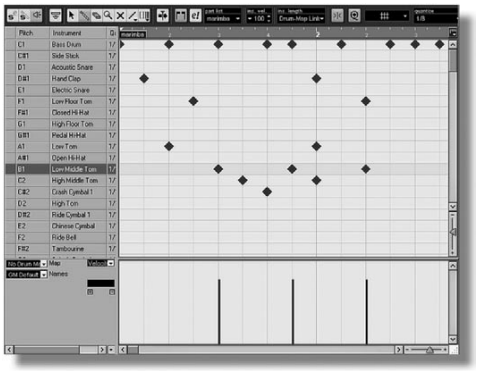
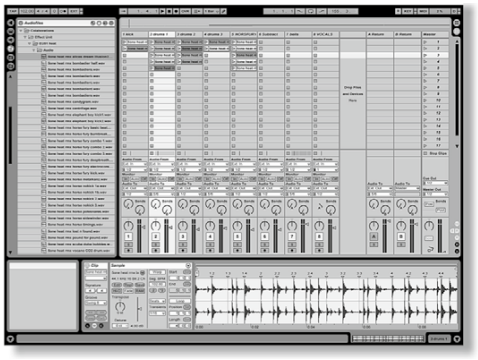
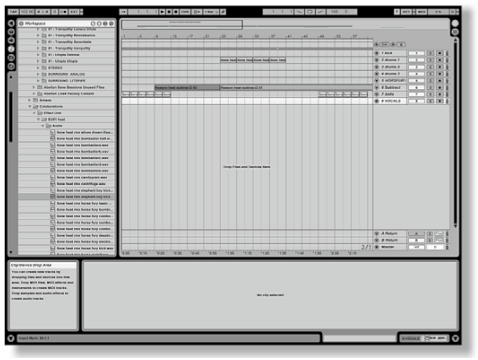


FIGURE 16.30
Steinberg Cubase/Nuendo drum edit window. (Courtesy of Steinberg Media Technologies GmbH, a division of Yamaha Corporation, www.steinberg.net.)

FIGURE 16.31
Steinberg's Sequel music software. (Courtesy of Steinberg Media Technologies GmbH, a division of Yamaha Corporation, www.steinberg.net.)



A. Arrangement view.



B. Session view.

FIGURE 16.32
Ableton live performance audio workstation. (Courtesy of Ableton, www.ableton.com.)

multitrack production interface. At their basic level, these programs differ conceptually from their traditional DAW counterpart, in that the pitch- and time-shift architecture is so variable and dynamic that even after the basic rhythmic, percussive and melodic grooves have been created, their tempo, track patterns, pitch, session key, etc. can be quickly and easily changed at any time. With the help of custom royalty-free loops (available from the manufacturer and/or third-party companies), users can quickly and easily experiment with setting up grooves, backing tracks, and creating a sonic ambience by simply dragging the loops into the program's main soundfile view where they can be arranged, edited, processed, saved, and exported.

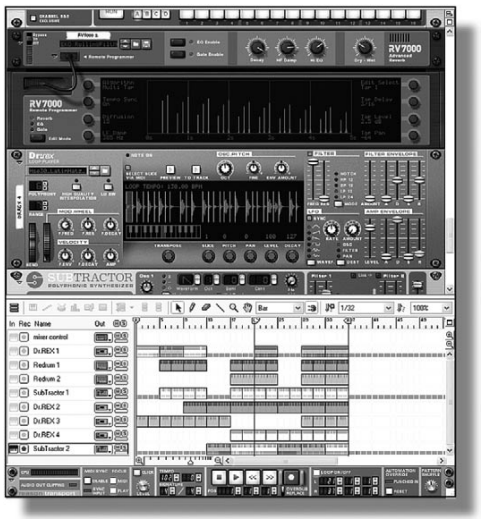


FIGURE 16.33
Reason music production environment. (Courtesy of Propellerheads software, www.propellerheads.se.)

One of the most interesting aspects of a loop-based editor is its ability to match the tempo of a specially programmed loop soundfile to the tempo of the current session. Amazingly enough, this process isn't that difficult to perform, as the program extracts the length, native tempo, and pitch information from the imported file's header and (using various digital time and/or pitch change techniques) adjusts the loop to fit the native time/pitch parameters of the current session. This means that loops of various tempos and musical keys can be automatically adjusted in length and pitch so as to fit in time with previously existing loops. These shifts in time to match a loop to the session's native tempo can actually be performed in a number of ways. For example, using basic DSP techniques to time-stretch and pitch-shift a recorded loop will often work well over a given plus-or-minus percentage range (which is often dependent on the quality of the program algorithms). Beyond this range, the loop will often begin to distort and become jittery. At such extremes, other playback algorithms and beat slice detection techniques can be used to make the loop sound more natural. For example, drums or percussion can be stretched in time by adding additional silence between the various hit points within the loop at precisely calculated intervals. In this way, the pitch will remain the same while the length is altered. Of course, such a loop would sound choppy and broken up when played on its own; however, when buried within a mix, it might work just fine. It's all up to you and the current musical context.

The software world doesn't actually hold the total patent on looping tools and toys; there are a number of groove keyboards and module boxes that are on the market. These systems, which range widely in sounds, functionality, and price, can offer up a wide range of unique sounds that can be quite useful laying a foundation under your production. In the past, getting a hardware grove tool to sync into a session could be time-consuming, frustrating, and problematic, taking time and tons of manual reading. However, with the advent of powerful time and pitch shift processing within most DAWs, the sounds from these hardware devices can be pulled into a session without too much trouble. For example, a single groove loop (or multiple loops) could be recorded into a DAW (at a bpm that's near to the session's tempo), edited, and then imported into the session, at which time the loop could be easily stretched into time sync, allowing it to be looped to your heart's content. Just remember, necessity is the mother of invention. Patience and creativity are probably your most important tools in the looping process.

If there's a software package that has gripped the hearts and minds of electronic musicians in the 21st century, it would have to be Reason from the folks at Propellerheads, Figure 16.33. Reason defies specific classification in that it's an overall music production environment that has many facets. For example, it includes a MIDI sequencer, as well as a wide range of software instrument modules which can be played, mixed, and combined in a comprehensive environment that can be controlled from any external keyboard and/or MIDI controller. Reason also includes a large number of signal processors that can be applied to any instrument or instrument group under full and easily controlled automation.

In essence, Reason is a combination of modeled representations of vintage analog synthesis gear, mixed with the latest digital synthesis and sampling technology. Combine these with a modular approach to signal and effects processing; add a generous amount of internal and remote mix and controller management (via an external MIDI controller); top this off with a quirky but powerful sequencer; and you have a software package that's powerful enough for top-flight production and convenient enough that you can build tracks from your laptop from your seat in a crowded plane. I know that it sounds like I read this from a sales brochure, but these are the basic facts of this program. When asked to explain Reason to others, I'm often at a loss as the basic structure is so open-ended and flexible that the program can be approached in as many ways as there are people who produce on it. That's not to say that Reason doesn't have a signature sound—it often does. However, it's a tool that can be either used on its own or in combination with other production instruments and tools.

Algorithmic Composition Programs. Algorithmic composition programs are interactive sequencers that directly interface with MIDI controllers or imported files to generate a performance in real-time, according to user-programmed computer parameters. In short, once you give it a few basic musical guidelines, it can act as a compositional robot to generate performances or musical parts on its own in order to help you gain new ideas for a song, create an automatic accompaniment, make improvisational exercises, create special performances, or just plain have fun.

This type of sequencer can be programmed to control the performance according to musical key, generated notes, basic order, chords, tempo, velocity, note density, rhythms, accents, etc. Alternatively, an existing standard MIDI file can be imported and further manipulated in real-time, according to new parameters that can be varied from a computer keyboard, mouse, or controller. Often such interactive sequencers will accept input from multiple players, allowing it to be performed as a collective jam. Once a composition has been satisfactorily generated, a standard MIDI file can be created and imported into any sequencer.

Patch Editors. The vast majority of MIDI instruments and devices store their internal patch data within RAM memory. Synths, samplers, or other devices contain information on how to configure oscillators, amplifiers, filters, tuning, and other presets in order to create a particular sound timbre or effect. In addition to controlling sound patch parameters, a unit's internal memory can also store such setup information as effects processor settings, keyboard splits, MIDI channel routing, controller assignments, etc.

Although these settings can be manually accessed from the device's panel controls, another (and sometimes more straightforward) way to gain real-time control over the parameters of an instrument or devices is through the use of a patch editor, Figure 16.34. A patch editor is a software package that's used to provide on-screen controls and graphic windows for emulating and varying an instrument's parameter controls in real-time.

Direct communication between a patch editor and the device's microprocessor commonly occurs through the use of MIDI SysEx messages. Almost all popular voice and setup editing packages include provisions for receiving and transmitting bulk patch data in this way. This makes it possible to save and organize large numbers of patch-data files, vary setting in real-time, and print out patch parameter settings.

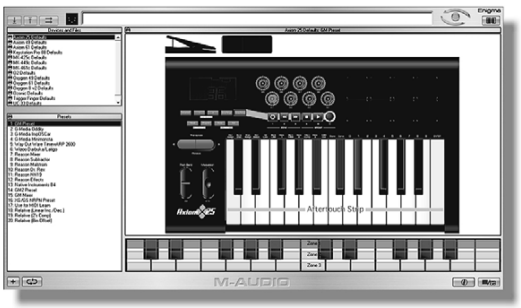


FIGURE 16.34
M-Audio Enigma Software Librarian and Editor. (Courtesy of M-Audio, a division of Avid Technology, Inc., www.m-audio.com.)

In addition to software editing packages, there are also hardware solutions for gaining quick and easy access to device parameters via SysEx. In recent years, MIDI data controllers, Figure 16.35, have sprung onto the market that can control a wide range of instruments and devices using data faders and soft buttons to vary patch, system, and performance parameters, in real-time. In many situations, these controllers can also be used to directly control the volume and mix parameters of a DAW.

Music-Printing Programs. In recent years, the field of transcribing musical scores onto paper has been strongly affected by computer, DAW, and MIDI technology. This process has been enhanced through the use of newer generations of software that make it possible for music notation data to be entered into a computer either manually (by placing the notes onto the screen via keyboard and/or by mouse movements) or via direct MIDI input. Once entered, these notes can be edited in an on-screen environment using a music printing program (or notation app within a DAW) that lets you change and configure a musical score or lead sheet using standard cut-and-paste edit techniques. In addition, most printing programs can play the various instruments in a MIDI system directly from the score. A final and important program feature is their ability to print out hard copies of a score or lead sheets in a wide number of print formats and styles.

These programs or DAW program apps, Figure 16.36, allow musical data to be entered into a computerized score in a number of manual and automated ways (often with varying degrees of complexity and ease). Although scores can be manually entered, most music-transcription programs will generally accept direct MIDI input, allowing a part to be played directly into a sequence. This can be done in real-time (by playing a MIDI instrument or finished sequence into the program), in step-time (entering

FIGURE 16.35
Mackie C4 plug-in and virtual instrument controller. (Courtesy of Loud Technologies, Inc., www.mackie.com.)

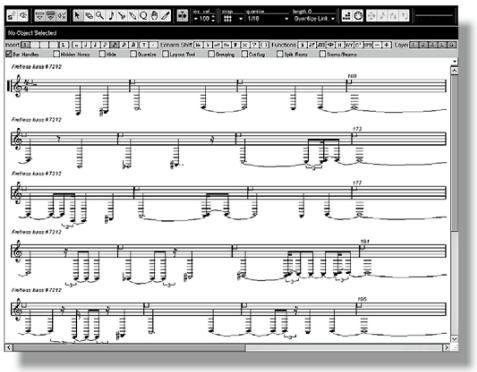


FIGURE 16.36
Score application within Steinberg's Nuendo DAW software. (Courtesy of Steinberg Media Technologies GmbH, a division of Yamaha Corporation, www.steinberg.net.)

the notes of a score one note at a time from a MIDI controller), or from an existing standard or program-specific MIDI file.

Another way to enter music into a score is through the use of an optical recognition program. These programs let you place sheet music or a printed score onto a standard flatbed scanner, scan the music into a program and then save the notes and general layout as a NIFF (notation interchange file format) file.

One of the biggest drawbacks to automatically entering a score via MIDI (either as a real-time performance or from a MIDI file) is the fact that music notation is a very interpretive art. "To err is human," and it's commonly this human feel that gives music its full range of expression. It is very difficult, however, for a program to properly interpret these minute yet important imperfections and place the notes into the score exactly as you want them. (For example, it might interpret a held quarter-note as either a dotted quarter-note or one that's tied to a thirty-second note.) Even though these computer algorithms are getting better at interpreting musical data and quantization can be used to tell a computer to round a note value to a specified length, a score will still often need to be manually edited to correct for misinterpretations.

MULTIMEDIA AND THE WEB

It's no secret that modern-day computers have gotten faster, sleeker, and sexier in their overall design. In addition to its ability to act as a multifunctional production workhorse, one of the crowning achievements of the modern computer is the degree of media and networking integration that has worked its way into our collective consciousness and become known as multimedia.

The combination of working and/or playing with multimedia has found its way into modern computer culture through the use of various hardware and software systems that work in a multitasking environment and combine to bring you a unified experience that seamlessly involves such media types as:

- Text
- Graphics
- Video
- Audio and music
- Computer animation
- MIDI

The obvious reason for integrating and creating these media types is the human desire to create content with the intention of sharing and communicating one's experiences with others. This has been done for centuries in the form of books and more recently by movies and television. In the here and now, the Web has been added to the communications list, in that it has created a vehicle that allows individuals (and corporate entities alike) to communicate a multimedia experience to millions and then allows each individual to manipulate that experience, learn from it, and even respond in an interactive fashion. The Web has indeed unlocked the potential for experiencing multimedia events and information in a way that makes each of us a participant, not just a passive spectator.

One of the unique advantages of MIDI, as it applies to multimedia, is the rich diversity of musical instruments and program styles that can be played back in real-time while requiring almost no overhead processing from the computer's CPU. This makes MIDI a perfect candidate for playing back soundtracks from multimedia games or over the Internet. It's interesting to note that MIDI has taken a back seat to digital audio as a serious music playback format for multimedia. Most likely, this is due to several factors, including:

1. A basic misunderstanding of the medium.
2. The fact that producing MIDI content requires a basic knowledge of music.
3. The frequent difficulty of synchronizing digital audio to MIDI in a multimedia environment.
4. The fact that soundcards often include poorly designed FM synthesizers (although most operating systems now include a higher-quality software synth).

Fortunately, an increasing number of software companies have taken up the banner of embedding MIDI within their media projects and have helped push MIDI a bit more into the Web and gaming mainstream. As a result, it's becoming more common for your PC to begin playing back a MIDI score on its own or perhaps in conjunction with a more data-intensive program or game.

Standard MIDI files

The accepted format for transmitting files or real-time MIDI information in multimedia (or between sequencers from different manufacturers) is the standard MIDI file. This file type (which is stored with a .mid or .smf extension) is used to distribute MIDI data, song, track, time signature, and tempo information to the general masses. Standard MIDI files can support both single and multichannel sequence data and can be loaded into, edited, and then directly saved from almost any sequencer package. When exporting a standard MIDI file, keep in mind that they come in two basic flavors: type 0 and type 1.

- Type 0 is used whenever all of the tracks in a sequence need to be compressed into a single MIDI track. All of the original channel messages still reside within that track; however, the data will have no definitive track assignments. This type might be the best choice when creating a MIDI sequence for the Internet (where the sequencer or MIDI player application might not know or care about dealing with multiple tracks).
- Type 1, on the other hand, will retain its original track information structure and can be imported into another sequencer type with its basic track information and assignments left intact.

General MIDI

One of the most interesting aspects of MIDI production is the absolute setup and patch uniqueness of each professional and even semipro project studio. In fact, no two studios will be alike (unless they've been specifically designed to be the same or there's some unlikely coincidence). Each artist will be unique in having his or her own favorite equipment, supporting hardware, favorite way of routing channels and tracks, and assigning patches. The fact that each system setup is unique and personal has placed MIDI at odds with the need for systems compatibility in the world of multimedia. For example, after importing a standard MIDI file over the Net and loading it into a sequencer, you might hear a song that's being played with a totally irrelevant set of sound patches (it might sound interesting, but it won't sound anything like it was originally intended). If the MIDI file is loaded into a new computer, the sequence will again sound completely different, with patches that are so irrelevant that the guitar track might sound like a bunch of machine-gun shots from the planet Gloob.

In order to eliminate (or at best reduce) the basic differences that exist between systems, a patch and settings standard known as General MIDI (GM) was created. In short, GM assigns a specific instrument patch to each of the 128 available program change numbers. Since all electronic instruments that conform to the GM format must use these patch assignments, placing GM program change commands at the header of each track will automatically configure the sequence to play with its originally intended sound. As such, no matter what sequencer is used to play the file back, as long as the receiving instrument conforms to the GM spec the sequence will be heard using its intended instrumentation. Tables 16.3 and 16.4 detail the program numbers and patch names that conform to the GM format (Table 16.3 for percussion and Table 16.4 for nonpercussion instruments). These patches include sounds that imitate synthesizers, ethnic instruments, and/or sound effects that have been derived from early Roland synth patch maps. Although the GM spec states that a synth must respond to all 16 MIDI channels, the first nine channels are reserved for instruments, while GM restricts the percussion track to MIDI channel 10.

MIDI-BASED SYNCHRONIZATION

Just as synchronization is routinely used in audio and video production, the wide acceptance of MIDI and digital audio within the various media has created the need for synchronization in project studio and midsized production environments. Devices such as MIDI sequencers, digital audio editors, effects devices, and digital mixing consoles make extensive use of synchronization and time code. However, advances in design have fashioned this technology into one that's much more cost-effective and easy-to-use—all through the use of MIDI. The following sections outline the various forms of synchronization that are often encountered in a MIDI-based production environment.

Simply stated, most current forms of synchronization use the MIDI protocol itself for the transmission of sync messages. These messages are transmitted along with other MIDI data over standard MIDI cables, with no need for additional or special connections.

Table 16.3 GM percussion instrument patch map (Channel 10).

35. Acoustic Bass Drum	59. Ride Cymbal 2
36. Bass Drum 1	60. Hi Bongo
37. Side Stick	61. Low Bongo
38. Acoustic Snare	62. Mute Hi Conga
39. Hand Clap	63. Open Hi Conga
40. Electric Snare	64. Low Conga
41. Low Floor Tom	65. High Timbale
42. Closed Hi-Hat	66. Low Timbale
43. High Floor Tom	67. High Agogo
44. Pedal Hi-Hat	68. Low Agogo
45. Low Tom	69. Cabasa
46. Open Hi-Hat	70. Maracas
47. Low Mid Tom	71. Short Whistle
48. Hi Mid Tom	72. Long Whistle
49. Crash Cymbal 1	73. Short Guiro
50. High Tom	74. Long Guiro
51. Ride Cymbal 1	75. Claves
52. Chinese Cymbal	76. Hi Wood Block
53. Ride Bell	77. Low Wood Block
54. Tambourine	78. Mute Cuica
55. Splash Cymbal	79. Open Cuica
56. Cowbell	80. Mute Triangle
57. Crash Cymbal 2	81. Open Triangle
58. Vibraslap	

Note: In contrast to Table 16.3, the numbers in Table 16.4 represent the percussion keynote numbers on a MIDI keyboard, not program change numbers.

MIDI real-time messages

Although no time code-based reference is implemented, it's important to know that MIDI has a built-in (and often transparent) protocol for synchronizing all of the tempo and timing elements of each MIDI device in a system to a master clock. This protocol operates by transmitting real-time messages to the various instruments and devices throughout the system. Although these relationships are usually automatically defined within a system setup, one MIDI device must be designated as the master device in order to provide the timing information to which all other slaved devices are locked. MIDI real-time messages consist of four basic types that are each 1 byte in length:

- **Timing clock**—A clock timing that's transmitted to all devices in the MIDI system at a rate of 24 pulses per quarter note (ppq). This method is used to improve the system's timing resolution and simplify timing when working in nonstandard meters (e.g., 3/8, 5/16, 5/32).
- **Start**—Upon receipt of a timing clock message, the start command instructs all connected devices to begin playing from the beginning of their internal sequences. Should a program be in midsequence, the start command repositions the sequence back to its beginning, at which point it begins to play.
- **Stop**—Upon the transmission of a MIDI stop command, all devices in the system stop at their current positions and wait for a message to follow.

Table 16.4 GM Non-percussion Instrument Patch Map with Program Change Numbers.

1. Acoustic Grand Piano	39. Synth Bass 1
2. Bright Acoustic Piano	40. Synth Bass 2
3. Electric Grand Piano	41. Violin
4. Honky-tonk Piano	42. Viola
5. Electric Piano 1	43. Cello
6. Electric Piano 2	44. Contrabass
7. Harpsichord	45. Tremolo Strings
8. Clavi	46. Pizzicato Strings
9. Celesta	47. Orchestral Harp
10. Glockenspiel	48. Timpani
10. Music Box	49. String Ensemble 1
12. Vibraphone	50. String Ensemble 2
13. Marimba	51. SynthStrings 1
14. Xylophone	52. SynthStrings 2
15. Tubular Bells	53. Choir Aahs
16. Dulcimer	54. Voice Oohs
17. Drawbar Organ	55. Synth Voice
18. Percussive Organ	56. Orchestra Hit
19. Rock Organ	57. Trumpet
20. Church Organ	58. Trombone
21. Reed Organ	59. Tuba
22. Accordion	60. Muted Trumpet
23. Harmonica	61. French Horn
24. Tango Accordion	62. Brass Section
25. Acoustic Guitar (nylon)	63. SynthBrass 1
26. Acoustic Guitar (steel)	64. SynthBrass 2
27. Electric Guitar (jazz)	65. Soprano Sax
28. Electric Guitar (clean)	66. Altto Sax
29. Electric Guitar (muted)	67. Tenor Sax
30. Overdriven Guitar	68. Baritone Sax
31. Distortion Guitar	69. Oboe
32. Guitar harmonics	70. English Horn
33. Acoustic Bass	71. Bassoon
34. Electric Bass (finger)	72. Clarinet
35. Electric Bass (pick)	73. Piccolo
36. Fretless Bass	74. Flute
37. Slap Bass 1	75. Recorder
38. Slap Bass 2	76. Pan Flute

(*Continued*)

Table 16.4 Continued.

77. Blown Bottle	103. FX 7 (echoes)
78. Shakuhachi	104. FX 8 (sci-fi)
79. Whistle	105. Sitar
80. Ocarina	106. Banjo
81. Lead 1 (square)	107. Shamisen
82. Lead 2 (sawtooth)	108. Koto
83. Lead 3 (calliope)	109. Kalimba
84. Lead 4 (chiff)	110. Bag pipe
85. Lead 5 (charang)	111. Fiddle
86. Lead 6 (voice)	112. Shanai
87. Lead 7 (fifths)	113. Tinkle Bell
88. Lead 8 (bass p lead)	114. Agogo
89. Pad 1 (new age)	115. Steel Drums
90. Pad 2 (warm)	116. Woodblock
91. Pad 3 (polysynth)	117. Taiko Drum
92. Pad 4 (choir)	118. Melodic Tom
93. Pad 5 (bowed)	119. Synth Drum
94. Pad 6 (metallic)	120. Reverse Cymbal
95. Pad 7 (halo)	121. Guitar Fret Noise
96. Pad 8 (sweep)	122. Breath Noise
97. FX 1 (rain)	123. Seashore
98. FX 2 (soundtrack)	124. Bird Tweet
99. FX 3 (crystal)	125. Telephone Ring
100. FX 4 (atmosphere)	126. Helicopter
101. FX 5 (brightness)	127. Applause
102. FX 6 (goblins)	128. Gunshot

- **Continue**—Following the receipt of a MIDI stop command, a MIDI continue message instructs all instruments and devices to resume playing from the precise point at which the sequence was stopped. Certain older MIDI devices (most notably drum machines) aren't capable of sending or responding to continue commands. In such a case, the user must either restart the sequence from its beginning or manually position the device to the correct measure.

Song position pointer

In addition to MIDI real-time messages, the Song Position Pointer (SPP) is a MIDI system common message that isn't commonly used in current-day production. Essentially, SPP keeps track of the current position in the song by noting how many measures have passed since the beginning of a sequence. Each pointer is expressed as multiples of six timing-clock messages and is equal to the value of a 16th note.

The song position pointer can synchronize a compatible sequencer or drum machine to an external source from any position within a song containing 1024 or fewer measures. Thus, when using SPP, it is possible for a sequencer to chase and lock to a multitrack tape from any measure point in a song.

Using such a MIDI/tape setup, a specialized sync tone is transmitted that encodes the sequencer's SPP messages and timing data directly onto tape as a modulated signal. Unlike SMPTE time code, the encoding method wasn't standardized between manufacturers. This lack of standardization prevents SPP data written by one device from being decoded by another device that uses an incompatible proprietary sync format.

Unlike SMPTE, where tempos can be easily varied by inserting a tempo change at a specific SMPTE time, once the SPP control track is committed to tape, the tape and sequence are locked into this predetermined tempo or tempo change map. SPP messages are usually transmitted only while the MIDI system is in the stop mode, in advance of other timing and MIDI continue messages. This is due to the relatively short time period that's needed to locate the slaved device to the correct measure position.

MIDI time code

MIDI time code (MTC) was developed to allow electronic musicians, project studios, video facilities, and virtually all other production environments to cost-effectively and easily translate time code into time-stamped messages that can be transmitted via MIDI. Created by Chris Meyer and Evan Brooks, MTC enables SMPTE-based time code to be distributed throughout the MIDI chain to devices or instruments that are capable of synchronizing to and executing MTC commands. MTC is an extension of MIDI 1.0, which makes use of existing message types that were either previously undefined or were being used for other non-conflicting purposes. Since most modern recording devices include MIDI in their design, there's often no need for external hardware when making direct connections. Simply chain the MIDI cables from the master to the appropriate slaves within the system (via physical cables, USB, or virtual internal routing). Although MTC uses a reasonably small percentage of MIDI's available bandwidth (about 7.68% at 30 fr/s), it's customary (but not necessary) to separate these lines from those that are communicating performance data when using MIDI cables. As with conventional SMPTE, only one master can exist within an MTC system, while any number of slaves can be assigned to follow, locate, and chase to the master's speed and position. Because MTC is easy to use and is often included free in many system and program designs, this technology has grown to become the most common and most straightforward way to lock together such devices as DAWs, modular digital multitracks, and MIDI sequencers, as well as analog and videotape machines (by using a MIDI interface that includes a SMPTE-to-MTC converter).

The MTC format can be divided into two parts:

- Time code.
- MIDI cueing.

The time code capabilities of MTC are relatively straightforward and allow devices to be synchronously locked or triggered to SMPTE time code. MIDI cueing is a format that informs a MIDI device of an upcoming event that's to be performed at a specific time (such as load, play, stop, punch in/out, reset). This protocol envisions the use of intelligent MIDI devices that can prepare for a specific event in advance and then execute the command on cue.

MTC is made up of three message types: quarter-frame messages, full messages, and MIDI cueing messages.

- **Quarter**—frame messages—These are transmitted only while the system is running in real or variable speed time, in either forward or reverse direction. True to its name, four quarter-frame messages are generated for each time code frame. Since eight quarter-frame messages are required to encode a full SMPTE address (in hours, minutes, seconds, and frames—00:00:00:00), the complete SMPTE address time is updated once every two frames. In other words, at 30 fps, 120 quarter-frame messages would be transmitted per second, while the full time code address would be updated 15 times in the same period. Each quarter frame message contains 2 bytes. The first byte is F1, the quarter-frame common header, while the second byte contains a nibble (four hits) that represents the message number (0 through 7) and a nibble for encoding the time field digit.
- **Full messages**—Quarter-frame messages are not sent in the fast-forward, rewind, or locate modes, as this would unnecessarily clog a MIDI data line. When the system is in any of these shuttle modes, a full message is used to encode a complete time code address within a single message. After a fast shuttle mode is entered, the system generates a full message and then places

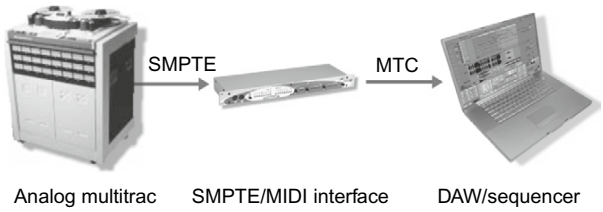


FIGURE 16.37
SMPTE time code can be easily converted to MTC (and vice versa) for distribution throughout a production system.

itself in a pause mode until the time-encoded slaves have located to the correct position. Once playback has resumed, MTC will again begin sending quarter-frame messages.

- **MIDI cueing messages**—MIDI cueing messages are designed to address individual devices or programs within a system. These 13-bit messages can be used to compile a cue or edit decision list, which in turn instructs one or more devices to play, punch in, load, stop, and so on at a specific time. Each instruction within a cueing message contains a unique number, time, name, type, and space for additional information. At the present time, only a small percentage of the possible 128 cueing event types has been defined.

SMPTE/MTC Conversion. Although MTC is commonly implemented within a software or hardware system itself (that's the functional and economic beauty of it), whenever a hardware device doesn't talk MTC (but only a flavor of the SMPTE protocol), a SMPTE-to-MIDI converter must be used, Figure 16.37. These conversion systems are available as stand-alone devices or as an integrated part of a multiport MIDI interface/patch bay/synchronizer system. Certain analog and digital multitrack systems include a built-in MTC port within their design, meaning that the machine can be synchronized to a DAW/sequencing system without the need for any additional hardware, beyond a MIDI interface.