CHAPTER e1

MIDI Basics

MIDI Internal Details

MIDI stands for Musical Instrument Digital Interface. It was developed by a consortium of musical product manufacturers to create a common language and protocol for musical software and hardware to communicate with each other. Before MIDI, the keyboard of one synthesizer brand could not send note data to another brand of synthesizer, nor was there a standard way to connect sound modules to the early computers of the day. I recall a session in 1982 when composer Jay Chattaway recorded the music for the movie *Vigilante* at my studio. The music was hard hitting and created entirely with early hardware synthesizers and samplers. For the week-long sessions Jay brought to my studio a very elaborate synthesizer setup, including a custom computer that played everything at once. It was a very complex and expensive system that took many hours to connect and get working.

MIDI changed all that. The original MIDI Specification 1.0 was published in 1982, and while it still carries the 1.0 designation, there have since been many additions and refinements. It's amazing how well this standard has held up for so long. You can connect a 1983 synthesizer to a modern digital audio workstation (DAW), and it will play music. Indeed, the endurance of MIDI is a testament to the power of standards. Even though the founding developers of the MIDI Manufacturer's Association (MMA) competed directly for sales of musical instrument products, they were able to agree on a standard that benefited them all. If only politicians would work together as amicably toward the common good.

As old as MIDI may be, it's still a valuable tool because it lets composers experiment for hours on end without paying musicians. MIDI data are much smaller than the audio they create because they're mostly 2- and 3-byte instructions. Compare this to CD-quality stereo audio that occupies 176,400 bytes per second. When I write a piece of music, I always make a MIDI mockup in SONAR first, even for pieces that will eventually be played by a full orchestra (heck—*especially* if they will be played by a real orchestra). This way I know exactly how all the parts will sound and fit together, and avoid being embarrassed on the first day of rehearsal. Imagine how much more productive Bach and Haydn might have been if they had access to the creative tools we enjoy today.

MIDI is also a great tool for practicing your instrument or just for making music to play along with for fun. Google will find MIDI files for almost any popular or classical music you want. You can load a MIDI file into your DAW program, mute the track containing your instrument, and play along. Further, a MIDI backing band never makes a mistake or plays out of tune, and it never complains about your taste in music either.

MIDI Hardware

The original MIDI communications spec was a hardware protocol that established the data format and connector types. A five-pin female DIN connector is used on the MIDI hardware, with corresponding male connectors at each end of the connecting cables. (DIN stands for Deutsche Industrie-Normen, a German standards organization.) Although MIDI was originally intended as a way for disparate products to communicate with one another, it's since been adapted by other industries. For example, MIDI is used to control concert and stage lighting systems and to synchronize digital and analog tape recorders to computer DAW programs via SMPTE as described in Chapter 6. MIDI is also used to send performance data from a hardware control surface to manipulate volume and panning and plug-in parameters in DAW software.

Figure e1.1 shows the standard arrangement of three MIDI jacks present on most keyboard synthesizers. The MIDI In jack accepts data from a computer or other controlling device to play the keyboard's own built-in sounds, and the MIDI Out sends data as you press keys or move the mod wheel, and so forth. The MIDI Thru port echoes whatever is received at the MIDI In jack to pass along data meant for other synthesizers. For example, a complex live performance synth rig might contain four or even more keyboard synths and sound modules, each playing a different set of sounds. So a master keyboard could send all the data to the first module in the chain, which passes that data on to all the others using their Thru connectors.

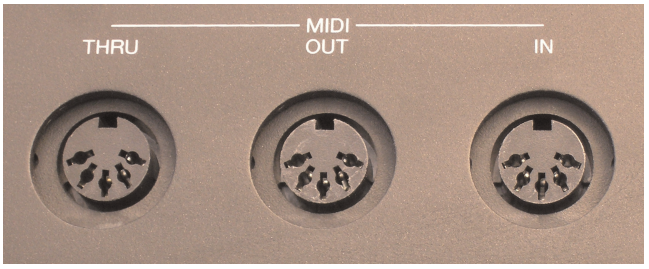

Figure e1.1: MIDI hardware uses a 5-pin DIN connector. Most MIDI devices include three ports for input, output, and pass-through (Thru) for chaining multiple devices.

MIDI Channels and Data

A single MIDI wire can send performance and other data to more than one synthesizer, or it can play several different voices at once within a single synth. In order to properly route data on a single wire to different synthesizers or voices, most MIDI data include a *channel number*. This number ranges between 0 and 15, representing channels 1 through 16, with the channel number embedded within the command data. For example, a note-on message to play middle C (note 60) with a velocity of 85 on Channel 3 sends three bytes of data one after the other, as shown in Table e1.1. MIDI channel numbers are zero-based, so the 2 in 92 means the note will play through Channel 3.

There are many types of MIDI data—note-on, note-off, pitch bend, sustain pedal up or down, and so forth—and most comprise three bytes as shown in Table e1.1. The first byte is treated as two half-bytes, sometimes called "nybbles" (or "nibbles"), each holding a range of 0 through 15 (decimal, or 0-F Hex). The higher (leftmost) nybble contains the command, and the lower nybble holds the channel number the message is intended for. The next two bytes contain the actual data, which usually ranges from 0 through 127 (7F Hex).

Some MIDI messages contain only two bytes when the data portion can be expressed in one byte. For example, after touch requires only a single byte, so the first byte is the command and channel number, and the second byte is the after touch data value. Likewise, a program change that selects a different voice for playback requires only two bytes. The program change command and channel number occupy one byte, and the new voice number is the second byte. Some MIDI messages require two bytes for each piece of data to express values larger than 127. The pitch bend wheel is one example, because 127 steps can't express enough in-between values to give adequate pitch resolution. So pitch bend data are sent as the command with channel number in one byte, followed by the lower value data byte, then the higher value data byte.

Every note sent by a MIDI keyboard includes a channel number, which usually defaults to Channel 1 unless you program the keyboard otherwise. If one keyboard is to play more than one voice, or control more than one physical sound module, you'll tell it to transmit on different channels when you want to play the other voices or access other sound modules.

Table e1.1: The MIDI Note-On Command.

Binary	Hex	Explanation
1001 0011	92	9 = Note-on command, 2 = Channel 3
0011 1100	3C	3C Hex = 60 = Note number
0101 0101	55	55 Hex = 85 = Velocity

Some keyboards have a split feature that sends bass notes out through one channel and higher notes out another. Likewise, each receiving device must be set to respond to the specific channels to avoid accidentally playing more than one voice or sound module at a time. Of course, if each synthesizer you'll play contains its own keyboard, this complication goes away.

Receiving devices can also be set to *omni mode*, in which case they'll respond to all incoming data no matter what channels it arrives on. This is common when recording to a MIDI track in a DAW program because it avoids those embarrassing head-scratching moments when you press a key and don't understand why you hear nothing. In omni mode, the incoming channel is ignored, so the receiving synthesizer will respond no matter which channel is specified. However, omni mode may not be honored when recording or playing a drum set through MIDI because many MIDI drum synths respond only to Channel 10.

Note that the term "channel" can be misleading, compared to TV and radio channels that are sent over different frequencies all at once. With MIDI data, the same wires or internal computer data paths are used for all channels. A header portion of the data identifies the channel number so the receiving device or program knows whether to honor or ignore that data. If you're familiar with data networking, the channel is similar to an IP address, only shorter.

MIDI Data Transmission

The original hardware MIDI standard uses a serial protocol, similar to the serial ports on older personal computers used to connect modems and early printers. Newer devices use USB, which is also a type of serial communication. Serial connections send data sequentially, one bit after the other, through a single pair of wires. So if you play a chord with eight notes at once, the notes are not sent all at the same time, nor are they received at the same time by the connected equipment.

Early (before USB) MIDI hardware uses a speed of 31,250 bits per second—called the *baud rate*—which sends about three bytes of data per millisecond. But most MIDI messages comprise at least three bytes each, so the data for that eight-note chord will be spread out over a span of about eight milliseconds by the time it arrives at the receiving synthesizer or sound module. For a solo piano performance, small delays like this are usually acceptable. But when sending MIDI data for a complete piece of music over a single MIDI cable, the accumulated delays for all the voices can be objectionable.

Imagine what happens on the downbeat of a typical pop tune as a new section begins. It's not uncommon for 20 or more notes to play all at once—the kick drum, a cymbal crash, five to ten piano notes, a bass note, another five to ten organ notes, and maybe all six notes

of a full guitar chord. Now, that 8-millisecond time span has expanded to 20 or even 30 milliseconds, and that will surely be noticed. The effect of hearing notes staggered over time is sometimes called *flamming*, after the "flam" drum technique where two hits are played in rapid succession.

Today, with software synthesizers and samplers that run within a DAW, MIDI messages are passed around inside the computer as fast as the computer can process them, so serial hardware delays are no longer a problem. But you still may occasionally want to move time-critical MIDI data between hardware synthesizers or between a hardware synth and a computer. I learned a great trick in the 1990s, when I wanted to copy all the songs in my Yamaha SY77 synthesizer to a MIDI sequencer program to edit the music more efficiently on my computer.

The SY77 is an all-in-one workstation that includes a 16-voice synthesizer using both samples and FM synthesis, several audio effects, plus built-in sequencing software to create and edit songs that play all 16 voices at once. Rather than play a song in real time on the SY77 while capturing its MIDI output on the computer, I set the SY77's tempo to the slowest allowed. I think that was 20 beats per minute (BPM). So the data for a song whose tempo is 120 BPM are sent from the synthesizer at a rate equivalent to six times faster than normal, thus reducing greatly the time offset between notes when played back on a computer at the correct tempo.

General MIDI

As valuable as MIDI was in the early days, the original 1.0 version didn't include standardized voice names and their corresponding program numbers. The computer could tell a synthesizer to switch to Patch number 14, but one model synth might play a grand piano, while the same patch number was a tuba on another brand. To solve this, in 1991 the MIDI spec was expanded to add General MIDI (GM), which defines a large number of voice names and their equivalent patch numbers. The standard General MIDI voice names and patch numbers are listed in Table e1.2. Note that some instrument makers consider these numbers as ranging from 1 to 128. So the tenth patch in the list will always be a glockenspiel, but it might be listed as Patch 10 instead of 9.

Of course, how different sound modules respond to the same note data varies, as do the basic sound qualities of each instrument. So a purely MIDI composition that sounds perfect when played through one synthesizer or sound module might sound very different when played through another brand or model, even if the intended instruments are correct. A soft drum hit might now be too soft or too loud, and a piano that sounded bright and clear when played at a medium velocity through one sound module might now sound muffled or brash at the same velocity on another sound module.

Table e1.2: Standard MIDI Patch Definitions.

Number	Patch Name	Number	Patch Name
0	Acoustic grand piano	64	Soprano sax
1	Bright acoustic piano	65	Alto sax
2	Electric grand piano	66	Tenor sax
3	Honkytonk piano	67	Baritone sax
4	Electric piano 1	68	Oboe
5	Electric piano 2	69	English horn
6	Harpsichord	70	Bassoon
7	Clavichord	71	Clarinet
8	Celesta	72	Piccolo
9	Glockenspiel	73	Flute
10	Music box	74	Recorder
11	Vibraphone	75	Pan flute
12	Marimba	76	Blown bottle
13	Xylophone	77	Shakuhachi
14	Tubular bells	78	Whistle
15	Dulcimer	79	Ocarina
16	Drawbar organ	80	Lead 1 (square)
17	Percussive organ	81	Lead 2 (sawtooth)
18	Rock organ	82	Lead 3 (calliope)
19	Church organ	83	Lead 4 (chiff)
20	Reed organ	84	Lead 5 (charang)
21	Accordion	85	Lead 6 (voice)
22	Harmonica	86	Lead 7 (fifths)
23	Accordion	87	Lead 8 (bass and lead)
24	Acoustic guitar (nylon)	88	Pad 1 (new age)
25	Acoustic guitar (steel)	89	Pad 2 (warm)
26	Electric guitar (jazz)	90	Pad 3 (polysynth)
27	Electric guitar (clean)	91	Pad 4 (choir)
28	Electric guitar (muted)	92	Pad 5 (bowed)
29	Overdriven guitar	93	Pad 6 (metallic)
30	Distortion guitar	94	Pad 7 (halo)
31	Guitar harmonics	95	Pad 8 (sweep)
32	Acoustic bass	96	FX 1 (rain)
33	Electric bass (finger)	97	FX 2 (soundtrack)
34	Electric bass (pick)	98	FX 3 (crystal)
35	Fretless bass	99	FX 4 (atmosphere)
36	Slap bass 1	100	FX 5 (brightness)
37	Slap bass 2	101	FX 6 (goblins)
38	Synth bass 1	102	FX 7 (echoes)
39	Synth bass 2	103	FX 8 (sci-fi)
40	Violin	104	Sitar
41	Viola	105	Banjo
42	Cello	106	Shamisen
43	Contrabass	107	Koto
44	Tremolo strings	108	Kalimba
45	Pizzicato strings	109	Bagpipe

(Continued)

Table e1.2: (Continued)

Number	Patch Name	Number	Patch Name
46	Orchestral harp	110	Fiddle
47	Timpani	111	Shanai
48	String ensemble 1	112	Tinkle bell
49	String ensemble 2	113	Agogo
50	Synth strings 1	114	Steel drums
51	Synth strings 2	115	Woodblock
52	Choir Aaahs	116	Taiko drum
53	Voice Oohs	117	Melodic tom
54	Synth voice	118	Synth drum
55	Orchestra hit	119	Reverse cymbal
56	Trumpet	120	Guitar fret noise
57	Trombone	121	Breath noise
58	Tuba	122	Seashore
59	Muted trumpet	123	Bird tweet
60	French horn	124	Telephone ring
61	Brass section	125	Helicopter
62	Synth brass 1	126	Applause
63	Synth brass 2	127	Gun shot

Fortunately, many modern samplers contain several different-sounding instruments within each type and offer more than one type of drum set to choose from. So Patch 000 in Bank 0 will play the default grand piano, while the same patch in Banks 1 and 2 contains additional pianos that sound different. Indeed, this is yet another terrific feature of MIDI: Unlike audio Wave files, MIDI data is easy to edit. All modern MIDI-capable DAW software lets you select a range of notes and either set new velocities or scale the existing velocities by some percentage. Scaling is useful to retain the variance within notes, where a musical phrase might get louder, then softer again. Or you could set all of the notes in a passage to the same velocity, not unlike applying limiting to an audio file.

In addition to defining standard patch names and numbers, General MIDI also established a standard set of note names and numbers for all of the drum sounds in a drum set. Table e1.3 shows the standard GM drum assignments that specify which drum sounds will play when those notes are struck on the keyboard or sent from a sequencer program via MIDI. This also helps to ensure compatibility between products from different vendors. As with GM voices, the basic tonal character of a given drum set, and how it responds to different velocities, can vary quite a lot. Further, some drum sets are programmed to cut off the sound when you release the key, while others continue sounds to completion even if the note length played is very short. But with GM, at least you know you'll get the correct instrument sound, if not the exact timbre.

Table e1.3: Standard GM Drum Assignments.

Note	Number Drum Sound	Note	Number Drum Sound
35	Bass drum 2	59	Ride cymbal 2
36	Bass drum 1	60	High bongo
37	Snare side stick	61	Low bongo
38	Snare drum 1	62	Muted high conga
39	Hand clap	63	Open high conga
40	Snare drum 2	64	Low conga
41	Low tom 2	65	High timbale
42	Closed hi-hat	66	Low timbale
43	Low tom 1	67	High agogo
44	Pedal hi-hat	68	Low agogo
45	Mid tom 2	69	Cabasa
46	Open hi-hat	70	Maracas
47	Mid tom 1	71	Short whistle
48	High tom 2	72	Long whistle
49	Crash cymbal	73	Short guiro
50	High tom	74	Long guiro
51	Ride cymbal 1	75	Claves
52	Chinese cymbal	76	High wood block
53	Ride bell	77	Low wood block
54	Tambourine	78	Mute cuica
55	Splash cymbal	79	Open cuica
56	Cowbell	80	Muted triangle
57	Crash cymbal 2	81	Open triangle
58	Vibraslap	82	Shaker

In addition to standards for voice names and numbers and drum sounds, MIDI also defines a standard set of *continuous controller* (CC) numbers. The standard continuous controllers are shown in Table e1.4, though not all are truly continuous. For example, the sustain pedal, CC #64, can be only On or Off. So any value of 64 or greater is considered as pushing the pedal down, and any value below 64 releases the pedal. These standard controllers are recognized by most modern software and hardware synthesizers, and some synthesizers recognize additional CC commands specific to features of that particular model.

Standard MIDI Files

Besides all the various data types, the MIDI standard also defines how MIDI computer files are organized. There are two basic types of MIDI file: Type 0 and Type 1. Type 0 MIDI files are an older format not used much anymore, though you'll occasionally find Type 0 files on the Internet when looking for songs to play along with. A Type 0 MIDI file doesn't distinguish between instrument tracks, lumping all of the data together onto a single track. In a Type 0 file, the only thing that distinguishes the data for one instrument from another

Table e1.4: Standard MIDI Continuous Controller Assignments.

CC#	Name
0	Bank select
1	Modulation wheel
2	Breath controller
4	Foot controller
5	Portamento time
6	Data entry
7	Main volume
10	Pan
11	Expression
32–63	LSB for controllers 0–31, when two bytes are needed
64	Sustain pedal
65	Portamento
66	Sostenuto pedal
67	Soft pedal
98	Nonregistered parameter number LSB
99	Nonregistered parameter number MSB
100	Registered parameter number LSB
101	Registered parameter number MSB
102–119	Undefined
121	Reset all controllers
122	Local control
123	All notes Off
124	Omni mode Off
125	Omni mode On
126	Mono mode On
127	Poly mode On

is the channel number. Fortunately, most modern MIDI software expands Type 0 files to multiple tracks automatically when you open them.

Type 1 MIDI files can contain multiple tracks, and that's the preferred format when saving a song as a MIDI file. Regardless, both MIDI file types contain every note and its channel number, velocity, and also a *time stamp* that specifies when the notes are to start and stop. Other MIDI data can also be embedded, such as on-the-fly program (voice) changes, continuous controller values, song tempo, and so forth. Those also have a time stamp to specify when the data should be sent.

MIDI Clock Resolution

Many DAW sequencer programs let you specify the time resolution of MIDI data, specified as some number of *pulses per quarter note*, abbreviated PPQ. These pulses are sometimes

Table e1.5: Pulse Spacing for MIDI Resolutions.

PPQ	Time Between Pulses
96	5.2 ms
240	2.1 ms
480	1.0 ms
960	0.5 ms

Table e1.6: Duration of Common Note Lengths at 240 PPQ.

Note Length	Number of Pulses
Whole note	960
Half note	480
Half note triplet	320
Quarter note	240
Quarter note triplet	160
Eighth note	120
Eighth note triplet	80
Sixteenth note	60
Sixteenth note triplet	40

called *clock ticks*. Either way, the PPQ resolution is usually set somewhere in the Options menu of the software and typically ranges from 96 PPQ through 960 PPQ. Since this MIDI time resolution is related to the length of a quarter note, which in turn depends on the current song tempo, it's not an absolute number of milliseconds. But it can be viewed as a form of quantization because notes you play between the clock pulses are moved in time to align with the nearest pulse. To put into perspective the amount of time resolution you can expect, Table e1.5 lists the equivalent number of milliseconds per clock pulse for different PPQ values at the common song tempo of 120 beats per minute.

Older hardware synthesizers often use a resolution of 96 PPQ, which at 5 ms is accurate enough for most applications. I use 240 PPQ for my MIDI projects because the divisions are easy to remember when I have to enter or edit note lengths and start times manually, and 2 ms is more than enough time resolution. To my thinking, using 960 PPQ is akin to recording audio at a sample rate of 192 KHz; it might seem like it should be more accurate, but in truth the improved time resolution is not likely audible. Even good musicians are unable to play with a timing accuracy better than about 20 to 30 milliseconds.[1] As a fun exercise to put this into proper perspective, try tapping your finger along with a 50 Hz square wave, and see how accurately you can hit every tenth cycle. Table e1.6 shows the

[1] *Movement-Related Feedback and Temporal Accuracy in Clarinet Performance*, by Caroline Palmer, Erik Koopmans, Janeen D. Loehr, and Christine Carter. McGill University, 2009.

duration in MIDI clock pulses of the most common note lengths at a resolution of 240 PPQ. If you ever enter or edit MIDI note data manually, it's handy to know the number of clock pulses for the standard note lengths. Even at "only" 240 PPQ, time increments can be as fine as 1/60 of a 1/16 note. It seems unlikely to me that any type of music would truly benefit from a higher resolution.

MIDI Minutiae

The Bank Select command is used when selecting patches on synthesizers that contain more than one bank for sounds. You won't usually need to deal with this data directly, but you may have to choose from among several available bank select methods in your software, depending on the brand of synthesizer you are controlling.

Most DAW and MIDI sequencer programs send an All Notes Off command out all 16 channels every time you press Stop to cut off any notes that might be still sounding. But sometimes you may get "stuck notes" anyway, so it pays to learn where the All Notes Off button is located in your MIDI software and hardware.

Besides the standard continuous controllers, General MIDI also defines *registered parameter numbers* (RPNs) and *nonregistered parameter numbers* (NRPNs). There are a few standard registered parameters, such as the data sequence that adjusts the pitch bend range to other than the usual plus/minus two musical half-steps. NRPNs are used to control other aspects of a synthesizer's behavior that are unique to a certain brand or model and thus don't fall under the standard CC definitions. The specific sequence of data needed to enable a nonstandard feature on a given synthesizer should be listed in the owner's manual.

Earlier I mentioned that some MIDI data require two bytes of data in order to express a sufficiently large range of values such as pitch bend. In that case, two bytes are sent one after the other, with the lower byte value first. When two bytes are treated as a single larger value, they're called the *least significant byte* (LSB) and *most significant byte* (MSB).

You can buy hardware devices to split one MIDI signal to two or more outputs or to merge two or more inputs to a single MIDI signal. A MIDI splitter is easy to build; it simply echoes the incoming data through multiple outputs. But a MIDI merger is much more complex because it has to interpret all the incoming messages and combine them sequentially without splitting up data that span multiple bytes. For example, if one input receives a three-byte note-on command at the same time another input receives a two-byte command to switch from a sax to a flute voice, the merger must output one complete command before beginning to send the other. Otherwise, the data would be scrambled with a command and channel number from one device, followed by data associated with another command from a different device.

Finally, MIDI allows transferring blocks of data of any size for any purpose using a method called *Sysex*, which stands for *system exclusive*. Sysex is useful because it can store and recall patch information for older pre−General MIDI synthesizers that use a proprietary format. I've also used Sysex many times to back up custom settings for MIDI devices to avoid having to enter them all over again if the internal battery fails.

Playing and Editing MIDI

Most MIDI-capable DAW programs work more or less the same, or at least have the same basic feature set. Again, I'll use SONAR for my examples because it's a full-featured program that I'm most familiar with. A program that records and edits MIDI data is often called a *sequencer*, though the lines have become blurred over the years now that many DAW programs can handle MIDI tracks as well as Wave file audio. Chapter 7 showed how audio clips can be looped, split, and slip-edited, and MIDI data can be manipulated in the same way. You can't apply cross-fades to MIDI data, but you can manipulate it in many more ways than audio files.

Because MIDI is data rather than actual audio, it's easy to change note pitches, adjust their length and velocity, and make subtle changes in phrasing by varying their start times. There are also MIDI plug-in effects that work in a similar way as audio effects. The MIDI arpeggiator shown in Chapter 14 operates as a plug-in, but there are also compressors that work by varying MIDI note-on velocity, echo and transpose (pitch shift) effects, and even chord analyzers that tell you what chord and its inversion is playing at a given moment.

MIDI tracks in SONAR include a *key offset* field to transpose notes higher or lower, without actually changing the data. This is useful when writing music for transposing instruments such as the clarinet and French horn. You can write and edit your music in the natural key for that instrument, yet have it play at normal concert pitch. But most MIDI sequencer software also offers destructive editing to change note data permanently. For example, you can *quantize* notes to start at uniform 1/8 or 1/16 note boundaries. This can often improve musical timing, and it is especially helpful for musicians who are not expert players. Then again, real music always varies at least a little, so quantizing can equally rob a performance of its human qualities. Most sequencer programs let you specify the amount of quantization to apply to bring the start times of errant notes *nearer* to the closest time boundary, rather than forcing them to an exactly uniform start time. Many sequencers also offer a *humanizing* feature that intentionally moves note start times away from "the grid" to sound less robotic.

Another important MIDI feature is being able to record a performance at half speed or even slower, which again is useful for people who are not accomplished players. If you

set up the metronome in your software to play every eighth note instead of every quarter note, it's easier to keep an even tempo at very slow speeds. I'm not much of a keyboard player, but I have a good sense of timing and dynamics control. So I often play solos at full speed on the MIDI keyboard, paying attention to phrasing and how hard I hit the keys, but without worrying about the actual notes I play. After stabbing at a passage "with feeling," I can go back later and fix all the wrong notes. I find that this results in a more musical performance than step-entering notes one by one with a mouse or playing very slowly, which can lose the context and feel of the music.

Figure e1.2 shows the *Piano Roll* view in SONAR, and other software brands offer a similar type of screen for entering and editing MIDI note data. Every aspect of a note can be edited, including its start time, length, and velocity. Controller data can also be entered and edited in the areas at the bottom of the window. The screen can be zoomed horizontally and vertically to see as much or as little detail as needed.

When you need to see and work with an even finer level of detail, the Event List shown in Figure e1.3 displays every piece of MIDI data contained in the sequence. This includes not just musical notes and controllers, but also tempo changes and RPN and NRPN data. If your synthesizer requires a special sequence of bytes to enable a special feature, this is where you'll enter that data.

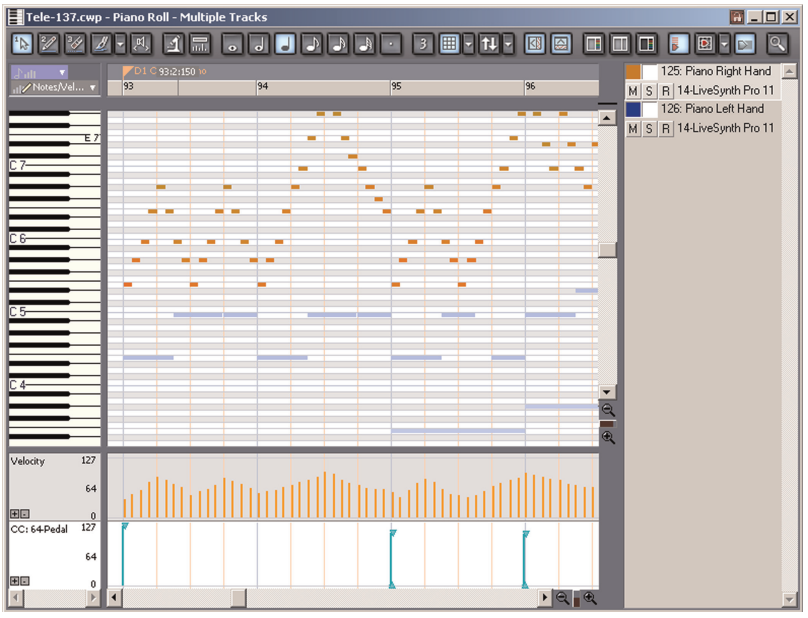

Figure e1.2: The Piano Roll window lets you enter and edit every property of MIDI notes and controller data.

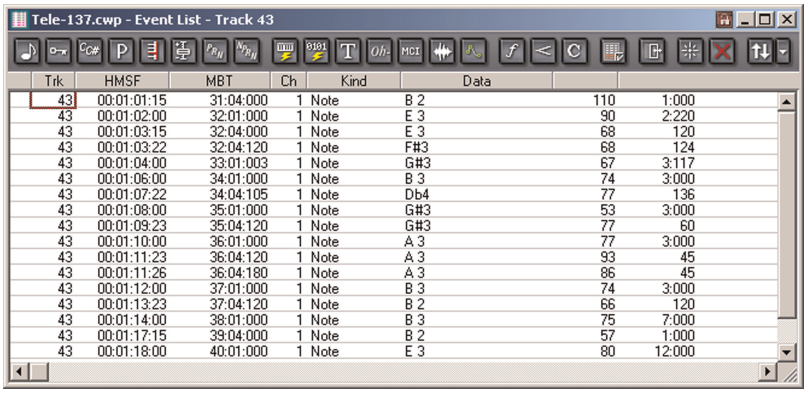

Figure e1.3: The Event List view offers even more detailed editing and data entry, showing every aspect of a MIDI project, including tempo changes, pitch bends, and other non-note parameters.

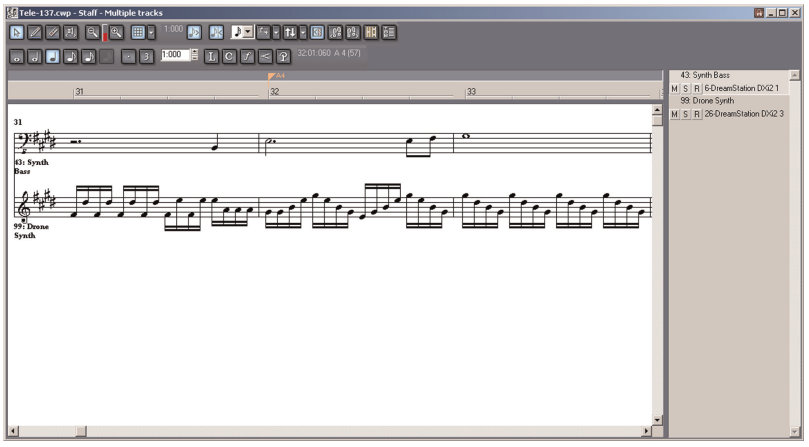

Figure e1.4: Most sequencers let you enter and manipulate MIDI data as musical notes in a Staff View, rather than as little bars on a grid.

Some musicians are more comfortable working with musical notes rather than computer data, and most sequencing software offers a *Staff View* similar to that shown in Figure e1.4. This type of screen is sometimes called *Notation View*. As with the Piano Roll view, you can edit notes to new pitches and start times, and even insert song lyrics and common dynamics symbols. The music display ability of most MIDI sequencers falls short of dedicated notation software, but many sequencer programs are capable of creating perfectly useable printed music and lead sheets. Some even include standard guitar symbols for all the popular chord types.

The "midi_editing" video shows an overview of MIDI editing basics, using the solo cello and piano section from my *Tele-Vision* music video as a demo. The piano track is entirely MIDI, which I created partly by playing notes on a keyboard and partly by entering and copying notes one at a time with a mouse. But the cello is my live performance; it's not a sampled cello!

Summary

This chapter covers MIDI internal details in depth, including hardware protocols, and data formats and their use of channels. The standard General MIDI instruments and drum notes were listed, along with miscellaneous MIDI tidbits such as MIDI file types, nonregistered parameters, and using Sysex to back up custom settings on MIDI hardware. Finally, a short video tutorial shows the basics of editing MIDI notes in a sequencer program.