MIDI *matters*

Sometimes a 'bit' of information can take a 'byte' out of MIDI hunger. In part three of our series, ET phones home with a message. *Text by Norman Weinberg.*

LAST MONTH WE learned that MIDI is nothing more than an electronic language that moves over a MIDI cable at over 31,000 bits per second. The purpose of this language is to communicate musical information from one device to another by turning the musical performance into numbers.

We also learned about status bytes (bytes beginning with the bit '1' that define a particular musical parameter) and data bytes (bytes which begin with the bit '0' and indicate an amount or level of the parameter defined by the status byte). This month, we'll get down and dirty into some of the specific MIDI commands that you will be using with your electronic drum setup.

MIDI Messages

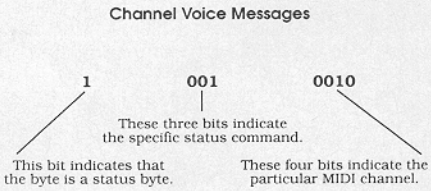
THE COMMANDS THAT make up the MIDI language are broken down into a few different categories – rather like the different parts of speech in the English language, where there are prepositions, nouns, verbs, and other classifications that describe how the words work within the syntax. In the MIDI language, words are classified into five main groups: Channel Voice Messages, Channel Mode Messages, System Common Messages, System Real-Time Messages, and System Exclusive Messages.

Let's take a look at Channel Voice Messages first, as these make up the majority of commands that describe your performance. If you remember back to last month's article, you know that there are 16 different MIDI channels. Channel Voice Messages get their name from the fact that these commands (status bytes) are channel specific, and because of this, travel over a single MIDI channel.

We can see how this works by looking at one of the most common MIDI messages, the Note On command. A Note On command is sent every time you strike an electronic drum pad or push the button on the front of a drum machine. This command simply means that a particular MIDI note number has been turned on. Every Note On command is followed by two different data bytes (there are 128 possible). The first data byte which follows a Note On command indicates the MIDI note that has been turned on, while the second data byte relates the velocity of that note.

Now that we know what a Note On command does, let's see exactly what makes it tick. By looking at Example #1, you can see how the different bits actually

Example #1



make up a typical Note On message. Because all status bytes must begin with the bit '1', that leaves only seven other bits that can define the status command. But, we use four of those bits to determine which MIDI channel the Channel Voice Message is on. The three bits in the middle make up the Note On command (001). The four bits at the far right designate the MIDI channel that this Note On command is intended for. In the example, the bit combination of 0010 means MIDI channel three. Since there are four bits that make up the MIDI channel designation, there are sixteen possible MIDI channels. They are (from MIDI channel one to sixteen): 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, and 1111.

Let's take just a second and define the velocity portion of the Note On message. In most cases, velocity can be defined as volume (or if you prefer, amplitude). But

this is not always the case. MIDI instruments that measure velocity (all electronic sets and triggers, most drum machines, and almost all keyboards) are really measuring the velocity or speed at which the surface moves from not being depressed to being depressed. With a keyboard synthesizer, velocity is a measurement of how fast the particular key moves from key-up position to key-down position. Most electronic drum sets are really measuring an amount of vibration when the surface is struck. If you hit a pad harder, the surface is going to vibrate more and the trigger will send a larger velocity reading.

Most MIDI devices will interpret this measurement as volume, but some can be programmed to send or 'map' this velocity reading to another MIDI parameter. As an example, it is quite easy to program many MIDI instruments to read velocity as a change in pitch level, a change in timbre (tone color), or to change the output in the stereo field. On some of the more advanced instruments, you can map the velocity reading to several different parameters at one time.

Example #2 shows the raw MIDI data of a drum machine playing a bass drum

Example #2

Raw Midi Data

1001 0011	Status Byte - Note On, Midi Channel Three
0010 0100	Data Byte - Midi Note Number 36
0110 1110	Data Byte - Velocity of 110

sound at a fairly strong volume. Notice the status byte for the note on command, followed by the two data bytes for which note (most drum machines use MIDI note number 36 for the bass drum sound) and its velocity (110 in this example). The complete MIDI message that fully describes the Note On event takes three bytes to complete.

AFTER A NOTE On event, the MIDI device must also send a Note Off message. This command is the exact opposite of a Note On, as it describes which note is going to stop sounding. The three middle bits that make up this command are '000'. Again, the first bit must be '1' for the status classification, and the last four bits convey the channel number. Note Off commands are extremely important! Without them, you get the dreaded 'MIDI drone' (stuck notes to you). As long as you rememger to pay your electric bill, a MIDI device will continue to play a sound until it receives a Note Off command for the same note number!

Every Note Off status byte is also followed by two data bytes, the first defines the MIDI note number that is going to stop, and the second indicates another velocity reading. This time, the velocity is a measurement of how fast the surface moves from the key-down to the key-up position. I don't know of any MIDI percussion devices that use this velocity reading. All electronic drums simply send a medium level of 64 for this value. There are a few keyboard synths that use the Note Off velocity to change how the sound is going to die away. As an example, a low velocity reading might cause the note to die away gradually, while a higher velocity could force the note to stop quickly.

There are two types of aftertouch defined by MIDI. Poly Key Pressure (bits 010) is used to measure aftertouch from each individual key, while Mono Key Pressure (bits 101) sends an average reading from the entire keyboard. In order to define aftertouch, let's think about a woodwind instrument. After the player starts blowing through the instrument, there are several things that can be done with the embouchure (the way the mouth is held to the mouthpiece) to change the sound. He or she can add vibrato, make the pitch rise or fall, make it louder or softer, or even change the tone color. Aftertouch was added to the MIDI specification as a way to change the sound in between the Note On and the Note Off commands.

With Poly Key Pressure, two additional data bytes are needed. The first will identify the note that has the pressure and the second will define the pressure value. Since Mono Key Pressure is an average reading and doesn't specify a particular note, it only needs a single data byte to indicate the pressure level.

Most often, instruments that respond to aftertouch use it to control the timbre or add some sort of LFO (low frequency oscillator) to the sound which creates vibrato. The only percussion devices I know of which use aftertouch are the Akai-Roger Linn MPC60 drum machine and Sequential's Studio 440. On the MPC60, aftertouch is used in conjunction with the note repeat feature. By holding down the note repeat key and pressing a drum button, the drum sound is repeated and each note's velocity is determined by the pressure on the drum button. This way, you can quickly program a crescendo without having to enter each note individually – quite a handy feature! The Studio 440 goes a couple of steps further by allowing you to change the panning and pitch as well, in much the same manner.

PROGRAM CHANGE IS another often used MIDI command. With this status byte, you can tell another MIDI device which program to play. Just what is a program? Good question. Different manufacturers use different names to indicate a program. Patch, tone, preset, configuration, and performance are all different names for a program. To make it easy, programs are what you are changing when you try to get your synth to change from a flute sound to a tuba sound. With electronic drum sets, programs are the different combinations of sounds that you hear when you play your pads. Some drum machines use Program Change messages to indicate a pattern or a sequence on the unit. Program Change status bytes (bits 100) are followed by a single data byte which tells the receiving device which program to go to.

Remember that there are only 128 different data bytes in MIDI, so there are also 128 different programs available. When the publicity blurb on a machine tells you that there are 128 possible programs, that's not a feature, that's MIDI!

Pitch wheels have been used on synthesizers from almost the very beginning. Today, even the least expensive synths will have a pitch wheel. This wheel is used to bend the sound's pitch up or down, depending on which direction you turn it. Because the pitch wheel is such a popular feature, the MIDI specification gave it its own status byte using the three bits of 110. Pitch wheels are a type of 'controller'. This is a fancy MIDI term that means that a knob, wheel, lever, joy stick, pedal, switch, or anything else can control the sound in one way or another.

Controllers work in a slightly different way than most other Channel Voice Messages. Because there are only 128 different data bytes available in MIDI, the people who put together the first MIDI specification felt that some controllers may need to have more than 128 different levels to describe their action. Why do we need more than 128 levels? If you're going to use pitch bend over the range of an octave (12 half-steps), then each half step is going to be divided into about ten parts. Rather than sounding like a smooth glide or glissando, it would sound a little choppy. A higher resolution of data is needed.

How do we get 'high resolution data'? By sending additional data bytes to describe the wheel's activity. Following a Pitch Wheel Change status command are two data bytes. The first data byte is called the 'Least Significant Byte' (LSB) and indicates a fine-tuning level of the controller. The second data byte is called the 'Most Significant Byte' (MSB) and indicates a sort of rough or course level. In other words, between the MSB values of 85 and 86, there are 128 different fine-tuning levels. When these two bytes (the LSB and MSB) are combined, they form a seven-bit word which gives a resolution of 16,384 different positions. This is quite an improvement over 128, and should be a high enough resolution for just about anyone!

There is another Channel Voice Message which is used for all the other controllers. The Control Change message (bits 011) sends instructions in a slightly different way than the pitch wheel.

Following the status byte, there are two or three data bytes that make up the complete message.

The first data byte actually defines which controller is being changed. In the first MIDI specification, only a few of the 128 possible controllers were defined. Currently, there are over thirty, including the modulation controller (1), main volume (7), pan controller (10), sustain pedal (64), and phaser depth (97).

The second data byte is the MSB (notice that the order of the MSB and LSB bytes are reversed from the Pitch Wheel Change command), and the third byte is the LSB if it is needed.

You may be asking yourself why you need to know about all of these other Channel Voice Messages. After all, if no percussion controllers support Note Off velocity, why should you mess with it? Well, the main reason that MIDI exists is so that you, as a drummer, can play and control any other instrument that supports the MIDI standard. Just because your drum machine doesn't have a pitch wheel, that doesn't mean that you can't hook up a pedal to your electronic drum set and bend the sound of a cymbal crash! With MIDI, you are in control of all the various parameters that make up music. The more parameters you can control, the more creative and individual your performances will become.

Example #3 is a reference chart of all the Channel Voice Messages. You can use this to find out how many data bytes follow each of the different status commands.

Example #3

Channel Voice Messages

Status Byte	First Data Byte	Second Data Byte
Note Off 1000 nnnn	Note Number	Velocity Value
Note On 1001 nnnn	Note Number	Velocity Value
Poly Key Pressure 1010 nnnn	Note Number	Pressure Value
Control Change 1011 nnnn	Control Number	MSB/LSB Value
Program Change 1100 nnnn	Program Number	None Sent
Mono Key Pressure 1101 nnnn	Pressure Value	None Sent
Pitch Wheel Change 1110 nnnn	LSB Value	MSB Value

"nnnn" represents the four bits that determine the MIDI channel.

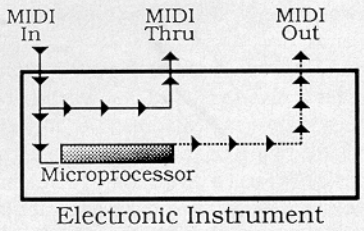
Transmission and Connections

NOW THAT WE'VE got the messages that send information about the performance, how do we get those messages from one MIDI device to another? The MIDI cable is a five-pin DIN type connector that carries the MIDI data stream. All MIDI devices have some sort of a MIDI port (a computer term for connecting jack) built into the case. There are three different types of MIDI ports that have very specific functions. The MIDI Out port sends signals away from the device. The MIDI data stream originates at the microprocessor (the unit's brain) and is sent out through this port. The MIDI In port brings the data stream into the microprocessor. It's really that simple, but there are a couple of important things to remember. First, MIDI signals move down a cable in only one direction. Second, every cable should connect from the master's MIDI Out port (sending device) to the slave's MIDI In port (receiving device). And, third, if you have your cables backwards, it won't work!

The third MIDI port is an optional port. It's called a MIDI Thru port and it works in a slightly different way. All the signals that are received by the MIDI In port are rolled around in the device and also sent out the MIDI Thru port. Confused? Take a look at Example #4. You can see that the signal from the MIDI In port is really going to

Example #4

MIDI Signal Paths

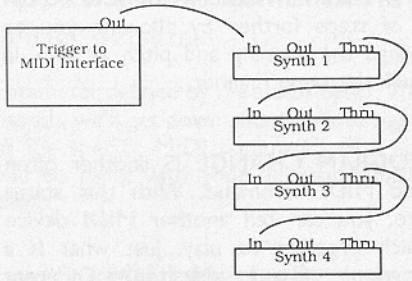


two different destinations: the micro-processor and the MIDI Thru port. The Out port, however, is sending signals that originate from the instrument's microprocessor. The signals origin is the main difference between MIDI Out and MIDI Thru. Signals that are sent by way of the MIDI Thru port began at some other device and were simply passed along by the receiving instrument.

By using the different MIDI ports in the proper way, it is possible to connect several instruments together in the same system. Example #5 shows the most

Example #5

Daisy Chain Connections

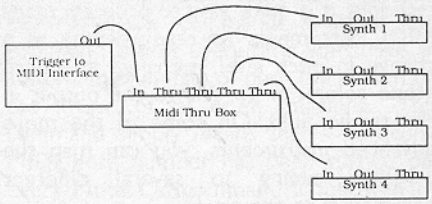


common method. Sometimes called a 'daisy chain', the MIDI signal originates at the master device (a trigger-to-MIDI interface) and passes from one slave's MIDI Thru port to another slave's MIDI In port. This is the type of connection that most people use. One problem, however, is that the synths in the example can never be used as a master device. Notice that there are no cable connections to their MIDI Out ports.

MIDI signals are subject to a small amount of distortion and noise when they are passed from the In port to the Thru port. After daisy chaining three MIDI devices, the signal may become so distorted that the receiving synths can't accurately interpret the data. The solution is a little contraption called a MIDI Thru Box. Example #6 shows a MIDI Thru box at work in a 'star' network. This additional

Example #6

Star Network Connections

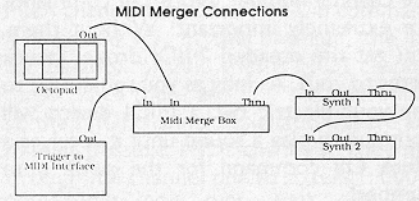


box can be a real lifesaver when your MIDI system begins to expand. Here, the signal from the trigger to MIDI interface goes to the thru box and is split into four more data streams. Each of the Thru ports contains an exact duplicate of the information received at the In port. By using this type of configuration, there is much less signal distortion. For more advanced and complicated MIDI systems, you can get a thru box which connects up to 24 MIDI devices. Try doing that with a daisy chain and see how far you get! I strongly recommend some sort of thru box if your MIDI systems consists of three or more devices. It will really make your life easier.

One of the big problems with MIDI is the fact that all instruments have only a single MIDI In port. This makes it impossible to have two master devices controlling a single slave. But, have no fear, you're not the first person who has wanted to do this. There are other little boxes called 'MIDI Mergers' that are built just to solve this problem. Example #7 shows you

Example #7

MIDI Merger Connections



how to set up a trigger-to-MIDI interface and an Octapad, so that both of their data streams are received by a single device.

Next month, Channel Mode Messages, System Exclusive Messages, and Timing Messages. ®