

# MIDI Basics

The word MIDI is an acronym. It stands for Musical Instrument Digital Interface. MIDI is the technology that allows electronic musical instruments to communicate with each other and with computers.

MIDI is not a device; there is nothing that a person can put his hands on and say, "This is the MIDI." Nor does MIDI communicate music or sound in the manner that CD,s tape recorders, and phonographs do. Strictly speaking MIDI is not even a language by which musical devices communicate although many people describe it that way.

MIDI is nothing more than a set of hardware and software specifications agreed upon by makers of electronic musical instruments and computers. When implemented in the manufacture of these devices, MIDI allows them to communicate performance commands with other MIDI capable devices. These performance commands are sent and received in the form of numeric codes.

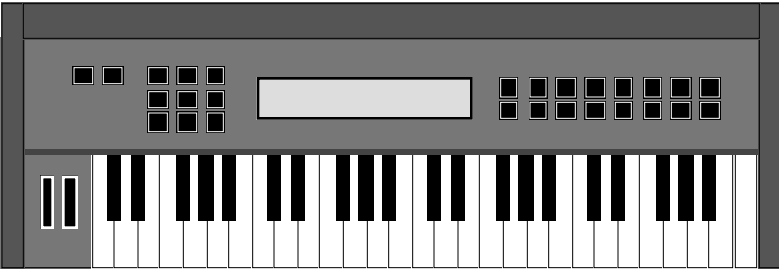
Many modern musical devices use digital technology to create music: audio CD's, digital tape recorders, even synthesizers. What makes MIDI different from these other technologies is that it uses numbers to represent *commands* to perform music rather than the sound itself. This allows for an immense savings in the quantity of numbers necessary to make a sound. A CD, for example, represents a second's worth of musical tone with over 44,000 numbers whereas a MIDI instrument can represent a tone with only six-- three to turn the note on and three to turn it off. This digital efficiency is the basis of MIDI's power and flexibility.

0 55 78 121 247 186 289 312 408 512 400 327 250 133 86 49 1 -1 -58 -82 -130 -255 -190 -250 -311 -414 -504 - 411 -  
 3: 12 400 327 250 133 86 49 1 -1 -58 -82 -130 -255 -190 -250 -  
 3: 7 186 289 312 408 512 400 327 250 133 86 49 1 -1 -58 -82 -  
 1: 3 -53 0 55 78 121 247 186 289 312 408 512 400 327 250 133  
 8: 1 -338 -261 -142 -78 -53 0 55 78 121 247 186 289 312 408  
 5: 0 -311 -414 -504 - 411 -338 -261 -142 -78 -53 0 55 78 121  
 247 186 289 312 408 512 400 327 250 133 86 49 1 -1 -58 -82 -130 -255 -190 -250 -311 -414 -504 - 411 -338 -261 -142  
 -78 -53 0 55 78 121 247 186 289 312 408 512 400 327 250 133 86 49 1 -1 -58 -82 -130 -255 -190 -250 -311 -414 -504 -  
 411 -338 -261 -142 -78 -53 0 55 78 121 247 186 289 312 408 512 400 327 250 133 86 49 1 -1 -58 -82 -130 -255 -190 -  
 250 -311 -414 -504 - 411 -338 -261 -142 -78 -53 0 55 78 121 247 186 289 312 408 512 400 327 250 133 86 49 1 -1 -58  
 -82 -130 -255 -190 -250 -311 -414 -504 - 411 -338 -261 -142 -78 -53 0 55 78 121 247 186 289 312 408 512 400 327  
 250 133 86 49 1 -1 -58 -82 -130 -255 -190 -250 -311 -414 -504 - 411 -338 -261 -142 -78 -53 0 55 78 121 247 186 289  
 312 408 512 400 327 250 133 86 49 1 -1 -58 -82 -130 -255 -190 -250 -



144 60 127 128 60 64

MIDI's use of numbers to represent performance commands rather than the music itself makes it much more efficient.

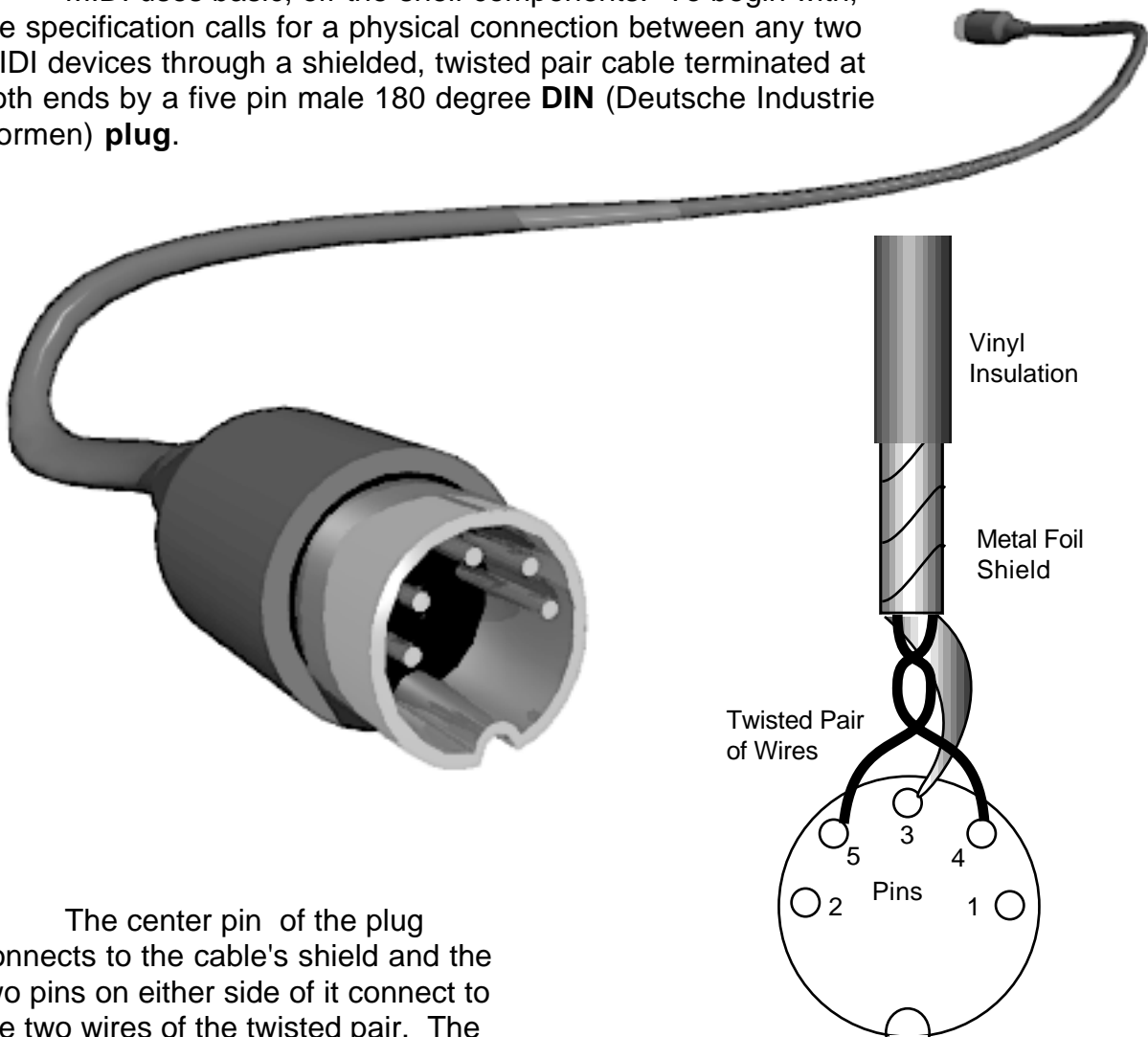


The MIDI specification was announced in 1983 at the winter NAMM show, and within a few years it gained such popularity that it became a required feature on any serious electronic musical instrument. Now, a dozen years later, despite numerous additions and some revision, the fact that it is still classified as MIDI 1.0 validates the robustness of its original concept.

Several other factors also contribute to its continued acceptance even though much more powerful technologies could be created today. One, from its inception it has been in the public domain; anyone can incorporate it into a commercial or non-commercial product without paying a license fee. Two, its technology is cheap; the specification calls for inexpensive, readily available hardware and easily implemented software. Two organizations now control MIDI's standardization and continued development: the **MMA** (MIDI Manufacturer's Association) and the **JMSC** (Japanese MIDI Standards Committee). Another organization, the **IMA** (International MIDI Association), oversees these groups and publishes the MIDI specifications.

## Hardware Specifications

MIDI uses basic, off-the-shelf components. To begin with, the specification calls for a physical connection between any two MIDI devices through a shielded, twisted pair cable terminated at both ends by a five pin male 180 degree **DIN** (Deutsche Industrie Normen) **plug**.

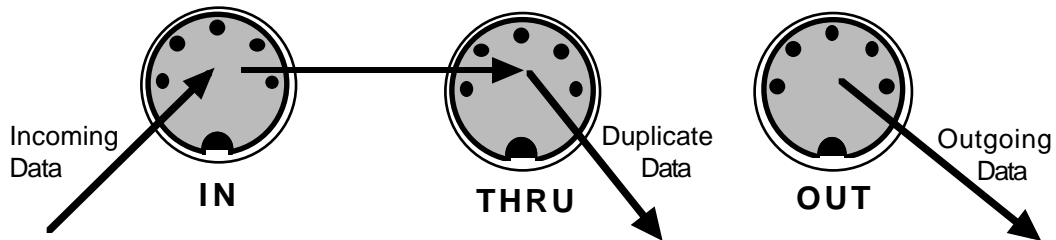


The center pin of the plug connects to the cable's shield and the two pins on either side of it connect to the two wires of the twisted pair. The outer two pins are left unconnected.

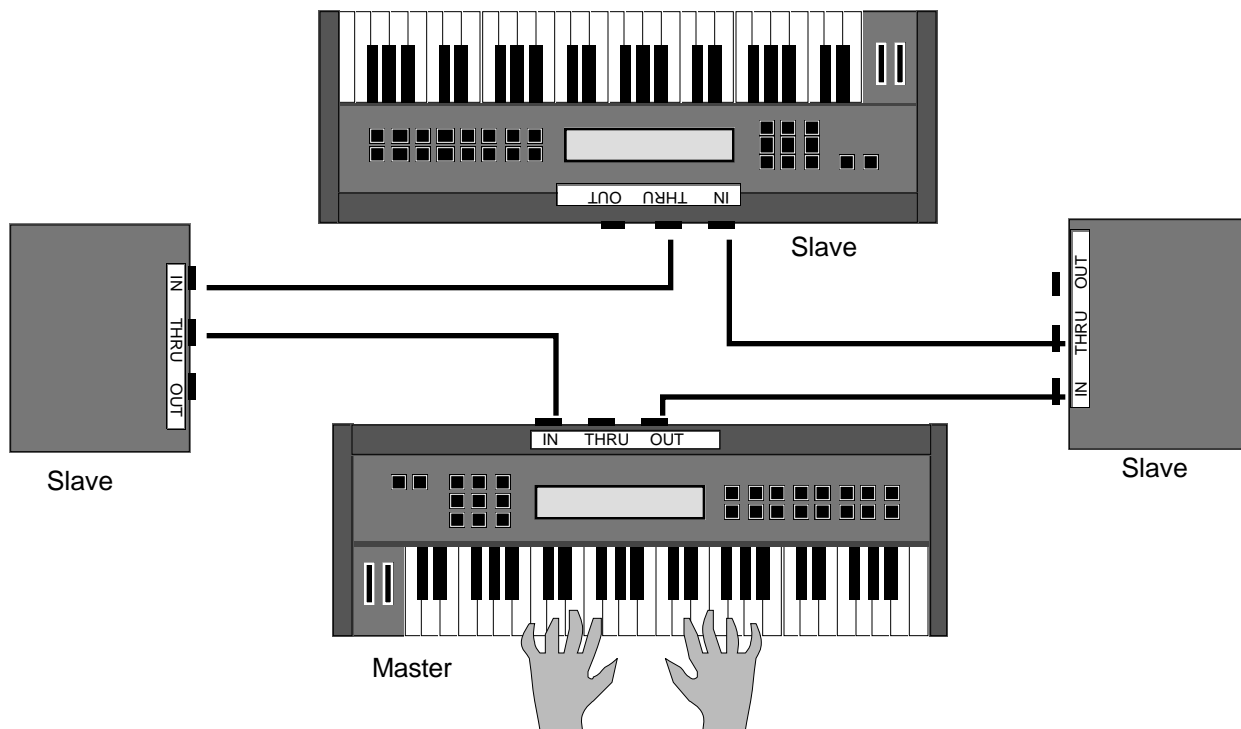
The cable between the two plugs can be any length under 50 feet. Longer than this risks data corruption primarily from electrical **induction**, interference generated by adjacent electrical devices.

One might well ask why standardize on a five pin plug if two pins are not used. The answer is that when the specification was proposed, five pin DIN plugs were less expensive and more readily available than other types-- an example of the philosophy that helped MIDI become so universal in such a short time.

The specification dictates that every MIDI capable device must have an input and/or an output port to match the cable plugs, that is a five hole 180 degree DIN socket. Most instruments have both IN and OUT, but some, designed only to receive or to transmit data, obviously need only one or the other. The **IN** port receives data from other connected devices, and the **OUT** port transmits data created within the device. An optional socket called the **THRU** port simply retransmits data that comes into the IN port on to the next connected device.

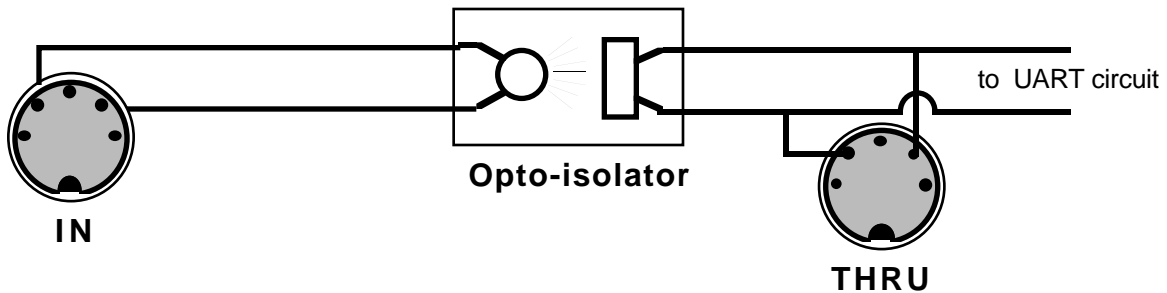


The THRU port theoretically allows any number of MIDI devices to be connected to the same bus in a configuration resembling, and often called, a **daisy chain**.

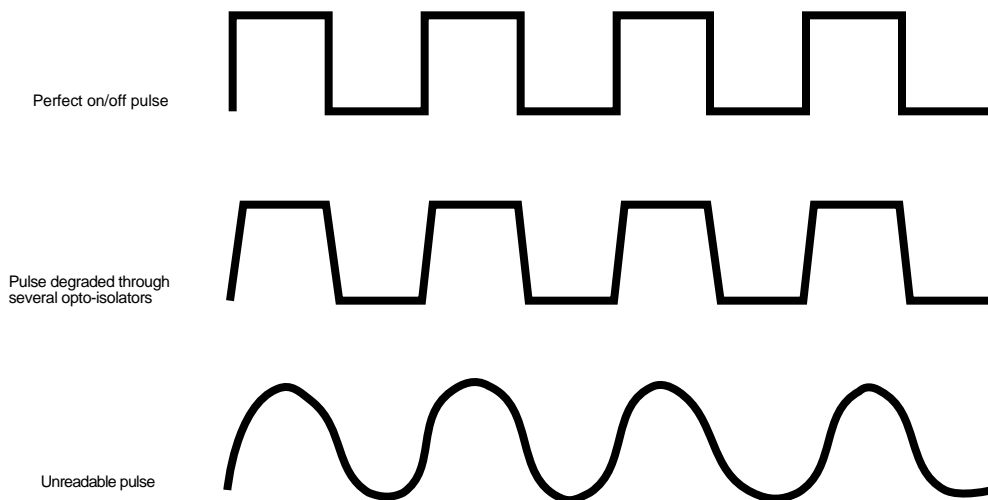


In any setup of multiple MIDI devices, the controlling instrument-- the one that the performer uses to initiate the MIDI data-- is named the **master** unit. All those that receive the data are called **slaves**.

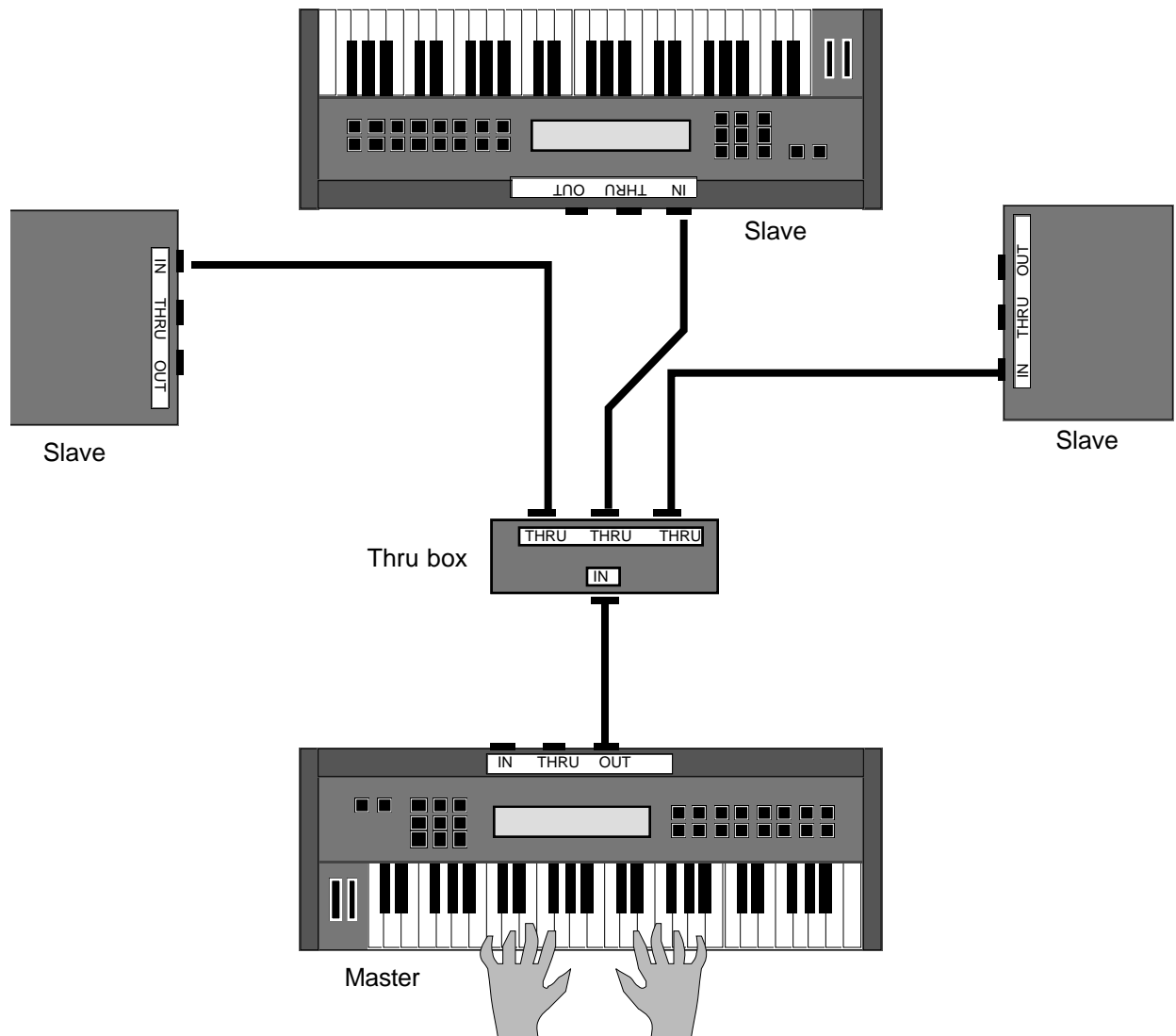
According to the spec, the IN port must be electrically separated from the rest of the instrument's electronics in order to prevent damage from possible incoming power surges. This is accomplished by an opto-isolator just behind the IN port. The **opto-isolator** is simply a circuit containing a light emitting diode and a photo-electric transistor. The diode turns on and off in response to the electrical pulses coming into the IN port. This light then stimulates the photoelectric transistor which acts as an on/off gate to electrical flow in the rest of the instrument's circuitry.



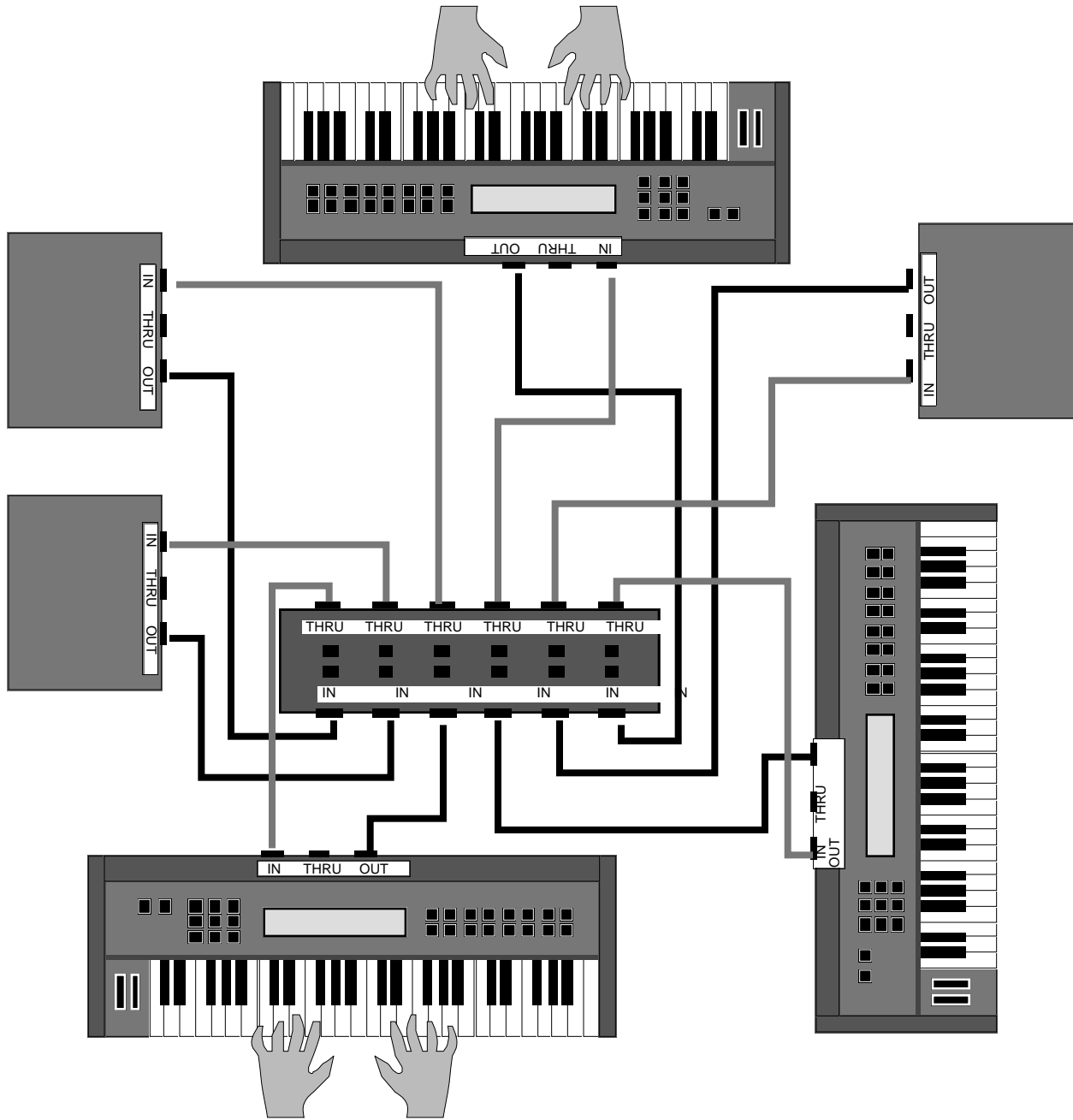
Opto-isolators are a safe way to convey electrical pulses without direct electrical connection, but they tend to degrade the swiftness with which an electrical pulse changes from on to off or from off to on. In a daisy chain of several MIDI devices this smoothing out of the binary signal, called **ramping**, becomes cumulatively more pronounced at each opto-isolator and so sets a practical limit to how many instruments can actually be hooked together in a chain of THRU port to IN port connections.



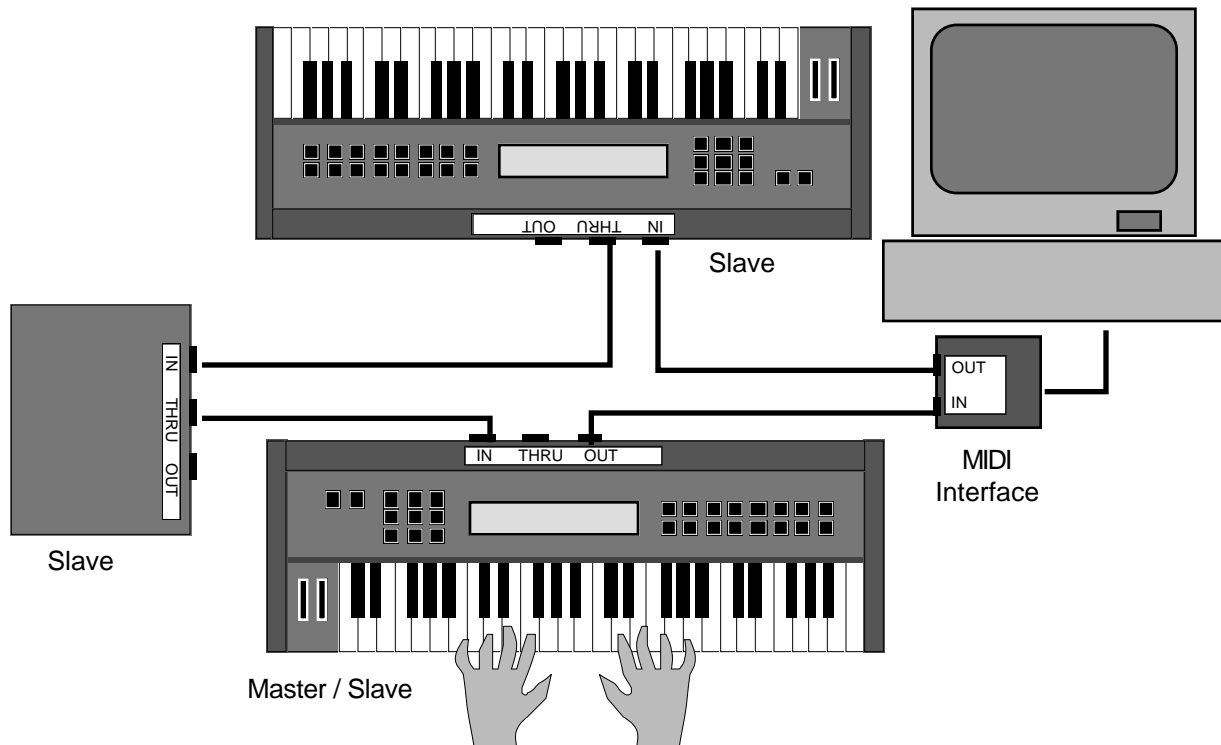
Although the daisy chain provides the easiest way to hook up multiple MIDI instruments, a different configuration called a **star network** avoids the problem of cumulative data degradation. In this arrangement the OUT port of the master unit is attached to a MIDI **THRU box**, a device which has one IN port and several THRU ports. Each slave is then connected to one of the box's THRU ports. This way each slave receives data that has passed through only one additional opto-isolator.



For the most flexible arrangement of numerous MIDI instruments a device known as a **MIDI patch bay** can be substituted for the THRU box. This device contains both IN ports and THRU ports for every instrument and allows the user to switch select which one should be the master and which the slaves. Some versions of this patch bay even allow for multiple masters at the same time; however, merging MIDI data without interlacing pulses from the data streams requires more intelligent circuitry.



In order to demonstrate IN, OUT, and THRU functions clearly, each of the previous set-up illustrations showed a layout only for real-time use-- one not able to record and play back MIDI data. Inserting a computer into any of them provides this capability but requires a little more information on how such a system functions. The following diagram shows a basic daisy chain set-up employing a computer.



The **MIDI interface** acts as a translator between the computer and the rest of the system matching each side's data speed and voltage requirements to the other. It usually connects to the computer through the serial port which allows two way flow of data over a single cable. Notice that the interface has no THRU port. This is because the OUT port normally serves that function when the computer is recording MIDI data. (Actually it allows the computer to read and retransmit the data as it records rather than simply duplicating the data directly from the IN port as other MIDI devices do, but the basic function is the same.) After data has been recorded and the computer is commanded to play it back, the OUT port serves its normal function.

Normally a master synth plays its own internal (local) sounds while simultaneously sending MIDI data. However, when it is connected in a complete daisy chain loop like that shown above, it is better to turn **local off**. If this is not done, each sound will be played twice, once locally and again through the MIDI chain. Although the player will usually not hear a double note, it halves the number of voices available.

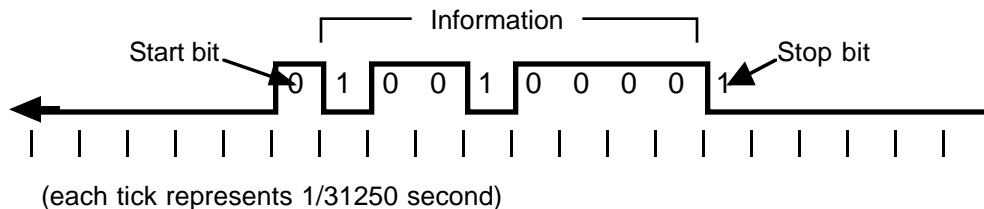


## Electrical Transmission

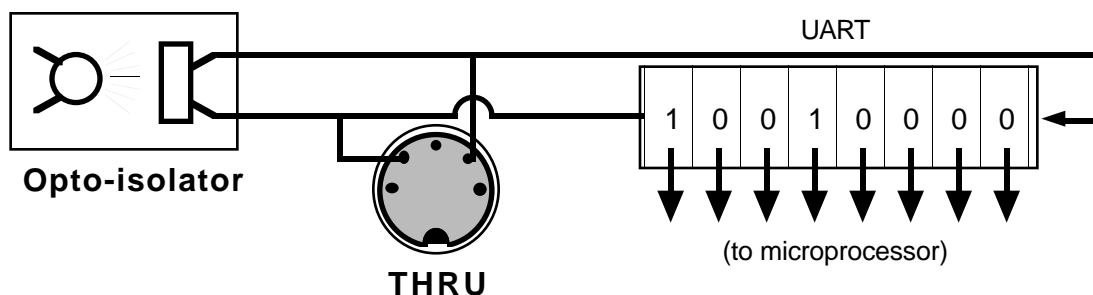
MIDI, like all digital technologies, works with binary numbers-- numbers represented by patterns of on or off electrical current. In the case of MIDI, current "on" is defined as be 5 volts electrical pressure at 5 milliamperes flow and is represented by a zero. Current "off" is, of course, no voltage and is represented by a 1. This seemingly backwards arrangement--one representing off and zero representing on-- is common in digital electronics and is known as **reverse logic**.

MIDI devices send and receive the patterns at **31,250 bps** (bits per second). The transmission is **serial**, that is all bits are transferred sequentially one after the other over a single line. It is also **asynchronous** which means that the sending and receiving timers do not have to be stepping together beforehand in order to communicate correctly. Again, the preference of serial asynchronous transmission over a faster parallel synchronous transmission and the choice of a relatively slow bps rate by today's standards reflect the desire by the early promoters of MIDI to keep its implementation as easy and inexpensive as possible.

The numeric codes that combine to represent MIDI performance commands are sent as eight bit bytes surrounded by a start bit of 0 and a stop bit of 1. Thus a full bit set of any MIDI byte is actually ten bits. (However, the start and stop bits are disregarded in discussing what information the MIDI byte is actually conveying.) Start and stop bits are required in asynchronous data transmission. The start bit serves as a signal for the receiving instrument to wake up and begin timing the incoming byte's pattern. The stop bit signals that the byte is complete.

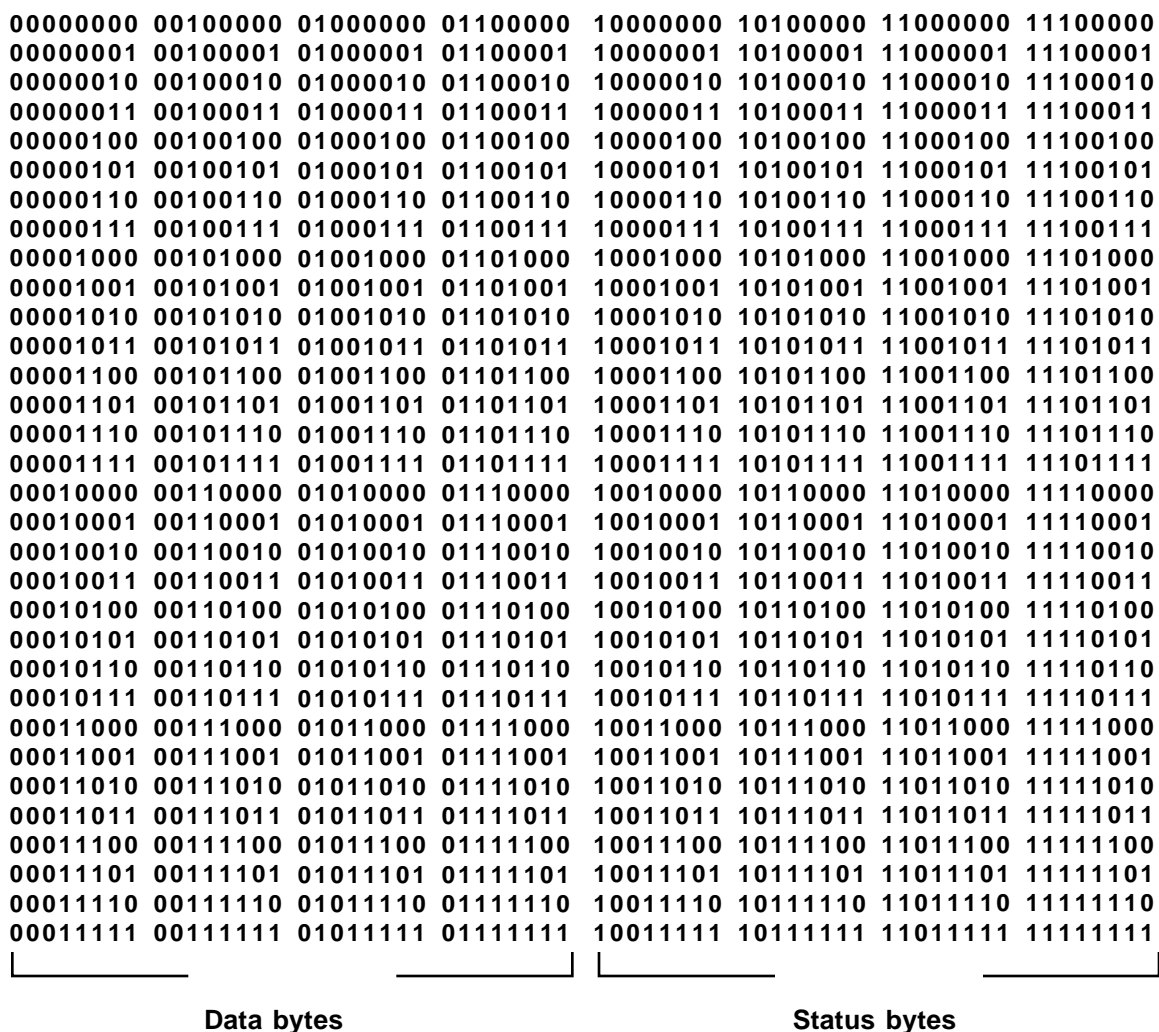


When the bits of a byte flow sequentially into the IN port and through the opto-isolator, they are collected one by one at 1/31250 second intervals in a circuit known as the **UART** (universal asynchronous receiver-transmitter). After the UART senses that it has captured all eight bits, it sends them together through a parallel bus to the microprocessor which compares the pattern to a library of recognized patterns and, if it finds a match, acts according to the stored directions.



As you know, an eight bit byte can be arranged into only 256 unique combinations. This number by itself would not permit a very large vocabulary of MIDI commands. Fortunately, the MIDI specification interprets not just single bytes but groups of bytes as commands. This multiplies the combinations, and the number of possible commands, exponentially.

To combine bytes meaningfully, MIDI actually starts by dividing the 256 combinations into two sets of 128 combinations. Those bytes which have an MSB of 1 (bytes representing decimal numbers 128 - 256) are called **status bytes**; they convey the actual command. Bytes with an MSB of 0 (bytes representing decimal numbers 0 to 127) are known as **data bytes**; they merely qualify or refine the command conveyed by the status byte.



Any group of bytes that can be interpreted as a meaningful MIDI performance command is known as a **MIDI message**. Every MIDI message must begin with a single status byte and may be followed by data bytes-- zero, one, two, or sometimes even thousands of them. The most common messages are composed of a status byte and one or two data bytes. A status byte and one data byte would yield 16,384

commands, and a status byte and two data bytes would allow 2,097,152 commands.

Putting all of the above technical data together reveals some interesting data about how fast MIDI can communicate. 31,250 bps divided by the number of bits in a byte's full bit set (10) divided by the typical number of bytes in a MIDI message (2 - 3) yields a normal communication speed of between 1000 and 1500 MIDI commands each second-- a lot of information by human standards!

## Overview of the MIDI Language

If we allow for some looseness of interpretation, it may be easier to think of a MIDI message as a command translated from a plain English sentence-- something like this: "Hey, channel one, turn on the note Middle C with your most forceful attack." In fact that command would look like the following in the MIDI language:

<b>10010001</b>	<b>00111100</b>	<b>01111111</b>
<small>Note on</small>	<small>Channel 1</small>	<small>Middle C (note #60)</small>
		<small>most forceful attack</small>

As in many foreign languages, the syntax seems somewhat strange to our eyes; however, all of the important elements of intelligible communication are present.

In MIDI, the status byte always represents the main part of the command, the subject ("channel one") and the predicate ("turn on the note"). The data bytes always represent qualifiers like adjectives, adverbs, prepositional phrases and the like.

We can categorize MIDI messages into two main classes: **channel messages** and **system messages**.

A **channel message** is meant to be received and acted upon only by those instruments in a MIDI setup which have been set to the same ID number as the incoming message. This is analogous to setting a television to receive the signal from only one channel even though signals from all channels are present in the cable. A channel message can be distinguished by the three bits of a status byte which follow the MSB. Of the eight possible combinations of these three bits, a channel message can be any of the following seven:

<b>000</b>	note off	<b>100</b>	program change
<b>001</b>	note on	<b>101</b>	channel pressure
<b>010</b>	poly pressure	<b>110</b>	pitch bend
<b>011</b>	control change		

Channel messages are often classified into two types: voice message and mode message. **Voice messages**, which include all those shown above, tell a specific sound generating element of a synthesizer what to do (turn on, turn off, change timbre, etc.). **Mode messages**, which are a subcategory of the control change message group above, tell a *synthesizer* how it should respond: monophonically, polyphonically, omni on (respond to data from all channels), or omni off (respond to data from a single channel).

The LSN of these channel messages (\_\_\_\_) indicates the number of the channel for which the information is meant. These four bits can represent any one of sixteen channels, 0 to 15.

_____	<b>0000</b>	= 0	_____	<b>1000</b>	= 8
_____	<b>0001</b>	= 1	_____	<b>1001</b>	= 9
_____	<b>0010</b>	= 2	_____	<b>1010</b>	= 10
_____	<b>0011</b>	= 3	_____	<b>1011</b>	= 11
_____	<b>0100</b>	= 4	_____	<b>1100</b>	= 12
_____	<b>0101</b>	= 5	_____	<b>1101</b>	= 13
_____	<b>0110</b>	= 6	_____	<b>1110</b>	= 14
_____	<b>0111</b>	= 7	_____	<b>1111</b>	= 15

(Manufacturers often label these channels 1 - 16 rather than 0 - 15.)

A **system message** is meant to be received by all instruments in a MIDI setup. It is indicated by the eighth combination of the three bits following a status byte's MSB:

\_\_**111**\_\_\_\_\_ = system message

Since a system message is meant for all instruments, the sixteen possible combinations of the LSN obviously no longer need to indicate the channel number. Instead they show what type of system message is being transmitted. These are arranged in the three categories shown on the following page:

**System Exclusive** messages:

\_\_\_\_\_ **0000** = manufacturer / universal sysex

**System Common** messages:

\_\_\_\_\_ **0001** = quarter frame

\_\_\_\_\_ **0010** = song position pointer

\_\_\_\_\_ **0011** = song select

\_\_\_\_\_ **0100** = (not yet defined)

\_\_\_\_\_ **0101** = (not yet defined)

\_\_\_\_\_ **0110** = tune request

\_\_\_\_\_ **0111** = end of exclusive

**System Real Time** Messages:

\_\_\_\_\_ **1000** = MIDI clock advance

\_\_\_\_\_ **1001** = (not yet defined)

\_\_\_\_\_ **1010** = start

\_\_\_\_\_ **1011** = continue

\_\_\_\_\_ **1100** = stop

\_\_\_\_\_ **1101** = (undefined)

\_\_\_\_\_ **1110** = active sensing

\_\_\_\_\_ **1111** = system reset

## Details of the MIDI Language

Following, status byte by status byte, is a more detailed command vocabulary and syntax:

### Channel messages:

#### Note Off, channels 0 - 15 (10000000 - 10001111)

**Note Off** is a three byte message comprised of a status byte followed by a data byte representing which note and a data byte representing how fast the note was released:

**10000000**    **00111100**    **01000000**  
Note off Channel 0      Middle C (#60)      Medium release (64)

The note number data byte can be any number from 0 to 127 (00000000 to 01111111) covering more than a ten octave scale from C (8 Hz.) to G (12,543 Hz). Middle C (261 Hz) is note number 60.

Although release velocity is meaningless on an acoustic keyboard, it can be set to control many effects on some MIDI synths.



### MIDI Note Numbers

#### Note On, channels 0 - 15 (10010000 - 10011111)

**Note On** is a three byte message comprised of the note on status byte followed by a data byte representing which note and a data byte representing the velocity with which the note was struck:

**10010000**    **00111100**    **01000000**  
Note on Channel 0      Middle C (#60)      Medium attack (64)

The note number data byte can be any number from 0 to 127 (00000000 to 01111111) covering more than a ten octave scale from C (8 Hz.) to G (12,543 Hz). Middle C (261 Hz) is note number 60.

The third byte, the velocity data byte, represents the time required for a synth key to travel from its rest position to a fully depressed position although this can be reversed. The fastest time assigns 127 to the byte which usually dictates a more accented sound. If a note on message has a velocity of 0, it is interpreted as a note off message. The least expensive MIDI synths often don't send or receive velocity data and play everything mezzo-forte.

## Poly Pressure, channels 0 - 15 (10100000 - 10101111)

**Polyphonic Pressure**, sometimes called **key aftertouch**, is a three byte message comprised of the poly pressure status byte followed by a data byte representing the note being affected and a data byte representing current finger pressure on the key:

**10100000**   **00111100**   **01000000**  
Poly pr.   Channel 0   Middle C (#60)   Medium pressure

The note identity data byte can be any number from 0 to 127. Middle C is note number 60. The pressure number data byte can also be any number between 0 and 127.

Poly pressure senses individual finger pressure changes and sends separate data for each key held down on a synthesizer. Implementation is relatively expensive and as a result is included only on more expensive instruments.

## Control Change, channels 0 - 15 (10110000 - 10111111)

In effect, the **Control Change** message opens another entire sub-page of secondary messages which allow more subtle control of a musical performance. They also allow for creative expansion of the MIDI spec by manufacturers and experimenters because many of them are not yet defined and some can be implemented in any way the maker sees fit.

Control change is a three byte message comprised of the control change status byte followed by a data byte representing which controller is being changed and another data byte representing the value of that change.

**10110000**   **00000111**   **01000000**  
C.change   Channel 0   Main volume   Mezzo forte (64)

The controller data byte (the second byte in the message) can be any number between 0 and 127. These numbers represent the controls classified below.

0 to 31 = **Continuous Controllers (MSB)**, A continuous controller represents any performance device that requires a wide, progressive range of settings like a volume pedal. or a fader. MIDI specifies the following:

- 0   **Bank select**
- 1   **Mod wheel**
- 2   **Breath controller**
- 3   Undefined
- 4   **Foot controller**

- 5 **Portamento time**
- 6 **Data entry**
- 7 **Main volume**
- 8 **Balance**
- 9 Undefined
- 10 **Pan**
- 11 **Expression**
- 12 **Effect control #1**
- 13 **Effect control #2**
- 14-15 Undefined
- 16 **General purpose #1**
- 17 **General purpose #2**
- 18 **General purpose #3**
- 19 **General purpose #4**
- 20-31 Undefined

32 to 63 = **Continuous Controllers (LSB)**, Because subtle performance nuances often require more than the 128 steps of resolution that the third byte of a control change message would provide, the above continuous controllers are duplicated by the following LSB group. The LSB controller divides each step of its equivalent MSB controller into 128 smaller parts thus giving a complete resolution of 16,384 steps (called **14 bit resolution**).

- |    |                          |       |                           |
|----|--------------------------|-------|---------------------------|
| 32 | <b>Bank select</b>       | 42    | <b>Pan</b>                |
| 33 | <b>Mod wheel</b>         | 43    | <b>Expression</b>         |
| 34 | <b>Breath controller</b> | 44    | <b>Effect control #1</b>  |
| 35 | Undefined                | 45    | <b>Effect control #2</b>  |
| 36 | <b>Foot controller</b>   | 46-47 | Undefined                 |
| 37 | <b>Portamento time</b>   | 48    | <b>General purpose #1</b> |
| 38 | <b>Data entry</b>        | 49    | <b>General purpose #2</b> |
| 39 | <b>Main volume</b>       | 50    | <b>General purpose #3</b> |
| 40 | <b>Balance</b>           | 51    | <b>General purpose #4</b> |
| 41 | Undefined                | 52-63 | Undefined                 |

64 - 69 = **Switch Controllers**, Switch controllers represent performance devices that need only two settings: on and off. This presents a potential interpretation problem to instrument manufacturers because the data byte used to represent the value of the control



change (the third byte of the message) has 128 settings. Officially MIDI now recognizes any number from 64 to 127 as "on" and 0 - 63 as "off." However, in order to remain compliant with older MIDI implementations, it is still recommended that 0 be used for "off" and 127 for "on." The switch controllers are:

- 64 **Sustain**
- 65 **Portamento**
- 66 **Sostenuto**
- 67 **Soft pedal**
- 68 **Legato**
- 69 **Sustain 2**

70 - 95 = **Single Byte Continuous Controllers**, These are additional continuous controllers that do not require the 14 bit resolution of controllers 1 - 63. They are:

- 70 **Timbre variations**
- 71 **Harmonic intensity**
- 72 **Release time**
- 73 **Attack time**
- 74-79 Undefined
- 80 **General purpose #5**
- 81 **General purpose #6**
- 82 **General purpose #7**
- 83 **General purpose #8**
- 84 **Portamento note**
- 85-90 Undefined
- 91 **Effects depth #1** (usually External effects)
- 92 **Effects depth #2** (usually Tremelo)
- 93 **Effects depth #3** (usually Chorus)
- 94 **Effects depth #4** (usually Detune)
- 95 **Effects depth #5** (usually Phaser)

96 - 101 = **Registered and Non-Registered Parameters**. These controllers in effect open a sub-sub-page of possible MIDI commands. The MSB and LSB of the non-registered parameters give the manufacturer 16,384 more numbers that he can assign to represent various functions of his synthesizer. These can be anything the maker wants. The registered parameters offer an equal number of possible controls; however they are reserved for future

expansion by the MIDI associations. Only five registered parameters are already assigned: **Pitch Bend Sensitivity** (LSB 00, MSB 00), **Fine Tuning** (LSB 01, MSB 00), **Coarse Tuning** (LSB 02, MSB 00), **Tuning Program** (LSB 03, MSB 00), and **Tuning Bank** (LSB 04, MSB 00).

Once a parameter is selected, a value is attached to it by sending either the Data entry control change bytes (cc #6 for MSB and cc #38 for LSB) or the data increment/decrement bytes (cc #96 and #97). The controllers are:

- 96 **Data increment**
- 97 **Data decrement**
- 98 **Non-registered parameter LSB**
- 99 **Non-registered parameter MSB**
- 100 **Registered parameter LSB**
- 101 **Registered parameter MSB**

102 - 119 = **Undefined Controllers**. More room for experimentation or future expansion.

120 - 127 = **Mode Controllers**. While all other channel messages are designed to give performance commands to individual synthesizer *voices*, these last seven control change data bytes are intended for the *synthesizer* as a whole. Each is explained below:

- 120 **All sound off**. Forces all sound generators to quit.
- 121 **Reset**. Re-initializes all controllers to their factory preset setting.
- 122 **Local control**. Turns on or off a synthesizer's ability to play its own sound generators
- 123 **All notes off**. A panic command which cuts off all sounds including stuck notes.
- 124 **Omni off**. Directs each MIDI channel to respond to messages designated for any channel.
- 125 **Omni on**. Directs each MIDI channel to respond only to messages designated for it.
- 126 **Mono**. Tells the synth to play one note at a time rather than notes simultaneously.
- 127 **Poly**. Directs the synth to allow notes to play

simultaneously.

Historically the meaningful combinations of the last four modes have been given the following numeric designations:

- Mode 1:** Omni on, poly
- Mode 2:** Omni on, mono
- Mode 3:** Omni off, poly
- Mode 4:** Omni off mono

### **Program Change, channels 0 - 15 (11000000 - 11001111)**

**Program Change** is a two byte message comprised of the program change status byte followed by a single byte representing the number of the **preset** (pre-programmed timbre) that the performer wishes the channel to use.

**11000000**      **00001000**  
P. change Channel 0      Preset #8

Note that this message does not actually change a timbre; it merely selects one of up to 128 available. This number may seem limiting to some users because many synths now have more than 128 timbres available in RAM at any one time. Some synths allow the user to redirect program numbers (for example the user sets the synth to call up preset # 160 when it receives program change #10. A newly added Control Change message (00) allows the user to select among multiple banks of presets.

Until recently there was no standardization of timbres-- preset #1 on one synth might be a piano sound while on another it might be a whistle. While this may not be a problem for the performer who uses the same set-up all the time, it presents a problem to those who want to exchange sequences or update equipment and still be assured that performances will sound essentially the same. The problem has been magnified in the past few years by the wide variety of MIDI compatible sound cards used in multimedia computers. With the introduction of the **General MIDI (GM)** extension to the MIDI spec in 1993, the situation, while still not perfect, has been helped.

The aspect of general MIDI that is pertinent to the program change message is that GM calls for standardized individual and group mapping of 128 timbres available through this message. This means that if preset number 1 sounds like a grand piano on one synthesizer, it will sound like a grand piano on another even if the tonal quality is somewhat different. Following is GM's recommended catalog of timbres and preset numbers:

<b>Keyboards:</b>	<b>Basses:</b>	<b>Reeds:</b>	<b>Synth effects:</b>
1 Grand piano	33 Acoustic bass	65 Soprano sax	97 Rain
2 Bright piano	34 Fingered bass	66 Alto sax	98 Soundtrack
3 Electric grand	35 Pick bass	67 Tenor sax	99 Crystal
4 Honky-tonk	36 Fretless	68 Bari sax	100 Atmosphere
5 Electric piano 1	37 Slap bass 1	69 Oboe	101 Brightness
6 Electric piano 2	38 Slap bass 2	70 English horn	102 Goblins
7 Harpsichord	39 Synth bass 1	71 Bassoon	103 Echo
8 Clavichord	40 Synth bass 2	72 Clarinet	104 Sci-Fi
<b>Keyboard</b>	<b>Strings:</b>	<b>Pipes:</b>	<b>Ethnic:</b>
<b>Percussion:</b>	41 Violin	73 Piccolo	105 Sitar
9 Celesta	42 Viola	74 Flute	106 Banjo
10 Glockenspeil	43 Cello	75 Recorder	107 Shamisen
11 Music box	44 Contrabass	76 Pan flute	108 Koto
12 Vibrafhone	45 Tremolo	77 Bottle	109 Kalimba
13 Marimba	46 Pizzicato	78 Shakuhachi	110 Bagpipe
14 Xylophone	47 Harp	79 Whistle	111 Fiddle
15 Tubular bells	48 Timpani	80 Ocarina	112 Shanai
16 Dulcimer	<b>Ensembles:</b>	<b>Synth lead:</b>	<b>Percussion:</b>
<b>Organs:</b>	49 String Ens. 1	81 Square lead	113 Tinkle bell
17 Drawbar organ	50 String Ens. 2	82 Saw lead	114 Agogo
18 Percussive organ	51 Synth str. 1	83 Calliope	115 Steel drum
19 Rock organ	52 Synth str. 2	84 Chiff	116 Woodblock
20 Church organ	53 Choir aahs	85 Charang	117 Taiko drum
21 Reed organ	54 Choir oohs	86 Voice	118 Melodic tom
22 Accordion	55 Synth voice	87 Fifths	119 Synth drum
23 Harmonica	56 Orchestra hit	88 Bass & lead	120 Reverse cymbal
24 Tango accordion	<b>Brass:</b>	<b>Pad:</b>	<b>Sound effects:</b>
<b>Guitars:</b>	57 Trumpet	89 New age pad	121 Guitar fret noise
25 Nylon guitar	58 Trombone	90 Warm pad	122 Breath noise
26 Steel guitar	59 Tuba	91 Polysynth pad	123 Seashore
27 Electric jazz	60 Mute tpt.	92 Choir	124 Bird tweet
28 Electric clean	61 French horn	93 Bowed pad	125 Telephone ring
29 Mute guitar	62 Brass section	94 Metallic pad	126 Helicopter
30 Overdrive	63 Synth brass 1	95 Halo	127 Applause
31 Distortion	64 Synth brass 2	96 Sweep	128 Gunshot
32 Harmonics			

In addition to mapping the above 127 presets, GM specifies the following note numbers and percussion sounds in mapping the drum kit to the synth keyboard:

35 Acoustic bass dr.	47 Low-mid tom	59 Ride cym 2	71 Short whistle
36 Bass drum 1	48 High-mid tom	60 High bongo	72 Long whistle
37 Side stick	49 Crash cym 1	61 Low bongo	73 Short guiro
38 Acoustic snare	50 High tom	62 Mute hi conga	74 Long guiro
39 Hand clap	51 Ride cym 1	63 Open hi conga	75 Claves
40 Electric snare	52 Chinese cym	64 Low conga	76 High woodbl.
41 Low-floor tom	53 Ride bell	65 High timbale	77 Low woodbl.
42 Closed hi-hat	54 Tamboutine	66 Low timbale	78 Mute cuica
43 High-floor tom	55 Splash cym	67 High agogo	79 Open cuica
44 Pedal hi-hat	56 Cowbell	68 Low agogo	80 Mute triangle
45 Low tom	57 Crash cym 2	69 Cabasa	81 Open trianpla
46 Open hi-hat	58 Vibraslap	70 Maraca	

Beyond these instrument mappings general MIDI gives several minimum hardware and software recommendations to manufacturers:

Untuned percussion use channel 10  
 All 16 MIDI channels supported  
 24 voice polyphony  
 Multitimbral operation with 16 simultaneous timbres  
 Dynamic voice allocation  
 Velocity sensitivity  
 Stereo output  
 Headphone output  
 Volume control  
 MIDI in port

### Channel Pressure, channels 0 - 15 (11010000 - 11011111)

**Channel Pressure**, sometimes called **channel aftertouch**, is a two byte message comprised of the channel pressure status byte followed by a data byte representing the current finger pressure on the entire keyboard:

**11010000**      **01000000**  
Ch. press. Channel 0      Medium pressure

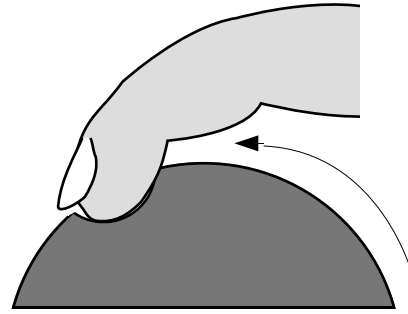
The pressure number data byte can be any number between 0 and 127.

Channel pressure is easier and less expensive to implement than polyphonic key pressure and as a result is found on many moderately priced synthesizers. Because it requires only two bytes it is also more economical in the amount of communication time it requires.

## Pitch Bend, channels 0 - 15 (11100000 - 11101111)

**Pitch Bend** is a three byte message that consists of the pitch bend status byte followed by an LSB data byte and an MSB data byte specifying the current position of the synthesizer's pitch wheel.

**11100000**      **01000000**      **01000000**  
Pitch b. Channel 0      Position LSB      Position MSB



Because the position is specified with two bytes, it can be shown with 14 bit accuracy or 16,384 steps even though many instruments use a coarser resolution (ignoring the less significant bits). The **center detente**, the central rest position, for pitch wheels is represented by the number 00000000 01000000 (8192) with higher or lower pitches represented by correspondingly higher or lower numbers.

Pitch wheel data does not directly specify pitch. The user can usually set the sensitivity of the tone generators to pitch wheel change. Thus a LSB/MSB combination of 16,384 might represent only a half-step above the normal pitch or it might represent a two octave change.

Even though MIDI is able to communicate over a thousand channel messages per second at 31,250 bps, there remains a danger that heavy data streams can overload its capacity leading to stuck notes and other glitches. Pitch bend, aftertouch, and continuous controllers all send information at the rate of 50 to 100 messages per second. Multiply that by several channels and it becomes apparent that it is indeed quite possible to exceed MIDI's **bandwidth**, that is, the number of bits per second it can carry. Fortunately, MIDI employs a data compaction technique known as **Running Status** on channel messages to help alleviate this potential problem. Simply stated, Running Status means that once the status byte of a channel message is sent, it remains in effect until a different status byte is received. (There is only one notable exception to this rule-- a MIDI clock status byte-- which will be discussed under system messages further on.)

Running Status saves the time required to retransmit the status byte for each similar event that sequentially follows. This compaction method works best for fast, continuously variable events like pitch wheel change, aftertouch, and continuous control changes. For singular events like program change and switch controllers it is not needed. Running Status is not used for system messages.

## System Messages:

### System Exclusive (11110000)

The **System Exclusive (sysex)** status byte is similar to the control change status byte in that it acts as a key to a sub-page of secondary messages which have many diverse functions. These secondary messages are indicated by the data bytes that immediately follow the sysex status byte.

System exclusive is a variable length message comprised of a sysex status byte followed by any number of data bytes and terminated by an end-of-exclusive (**EOX**) status byte (11110111).

**11110000 01000001 .... .... .... .... 11110111**  
System exclusive Roland ID# Message data bytes..... End of Exclusive

System exclusive messages are grouped into the following types identified by the second byte in the message:

0 to 124     **Manufacturers messages.** Manufacturers messages are designed to communicate only with designated brands and models of MIDI equipment in a mixed instrument MIDI setup. They can encode anything that the instrument maker wishes and that he will put in public domain. They typically are used to change the timbre characteristics of a synth preset.

From the inception of MIDI, the IMA has assigned a unique single byte number between 1 and 124 to each participating instrument maker. The first MIDI manufacturer, Sequential Circuits, was number 01. Some other early companies' ID's are: 67 = Yamaha, 65 = Roland, 15 = Ensoniq, 17 = Apple Computer, etc.

When the MIDI regulating organizations realized that they would have to accommodate many more than 124 manufacturers, they began to use a three byte system for later participants. The first byte for these newer makers is 00 with the following two bytes representing the LSB and MSB of a number between 0 and 16,384.

A typical sysex message is shown below in its usual hexadecimal form. This one directs an EMU Proteus 1 sound engine to create a grand piano sound as preset #1.

```

F0 18 04 00 01 00 00 41 00 63 00 6F 00 75 00 73
00 74 20 00 00 47 00 72 00 61 00 6E 00 64 00 7F
7F 7F 7F 7F 7F 00 00 00 00 00 00 00 00 7F 00 7F
00 7F 00 7F 00 01 00 00 00 00 00 00 00 69 00 00
00 00 00 00 00 7F 00 00 00 29 00 4D 00 00 00 12
7F 7F 00 7B 7F 00 00 00 00 7F 00 00 00 00 00 00
00 63 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 40 00 00 01 40 00 00 00 3C 00 00 00 00 00 00
00 01 00 3C 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 63 00 00 00 00 00 01 00 00 00 01 00 01
00 01 00 01 00 04 00 00 00 00 00 00 00 00 00 00
00 7F 00 40 00 40 00 40 00 40 00 40 00 00 00 01
00 07 00 02 00 05 00 06 00 08 00 09 00 01 00 11
00 01 00 00 00 00 00 00 00 00 00 00 00 01 00 00
00 00 00 20 00 7F 00 7F 00 7F 00 0C 00 0D 00 05
00 BC 00 00 00 00 00 1D F7

```

125           **Universal noncommercial messages.** The MIDI associations have set aside this number to denote messages for experimentation and research.

126           **Universal non-real time messages.** These messages are used for communicating information that is not time critical to the entire MIDI setup. After the 126 ID, a sub-ID byte follows to designate what type of information the message contains. Some examples of its uses:

- 01 - 03    = Sample dump information
- 04         = MIDI time code
- 05         = Sample dump extensions
- 06         = General information
- 07         = Standard MIDI File dump
- 08         = MIDI tuning standard
- 09         = General MIDI on/off

127           **Universal real time messages.** These messages are designed to relay information that is time critical to the entire MIDI setup. After the 127 ID, a sub-ID byte follows to designate what type of information the message contains. Some examples of its uses:

- 01         = MIDI time code
- 02         = MIDI Show Control



- 03 = Notation
- 04 = Device control
- 05 = Real time cueing
- 06 = MIDI machine control commands
- 07 = MIDI machine control responses
- 08 = MIDI tuning

### System Common Quarter Frame (11110001)

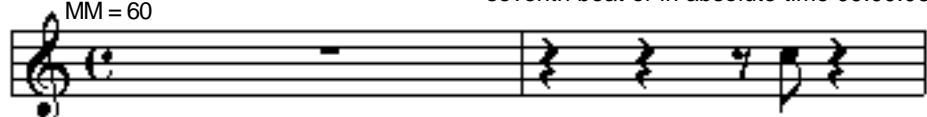
The Quarter Frame message was added to the MIDI specification in 1987 to allow the transmission of absolute time information through MIDI. **Absolute time** refers to the timing of events in hours, minutes, seconds, and parts of a second called **frames** after a given starting point. Described simply, a frame usually equals a 30th of a second although this varies for different uses.

Absolute time differs from the metronomic timing common in music. Metronomic or **Relative time** refers to the placement of events in tempo dependent units of measures, beats, and sub-beats after a given starting point. Absolute time is also called **SMPTE** time after the Society of Motion Picture and Television Engineers that adopted it as a standard in 1970. SMPTE time is usually shown in the following form:

**00:00:00:00**  
Hours      Minutes      Seconds      Frames

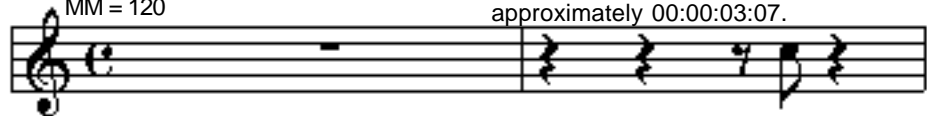
The difference between relative time and absolute or SMPTE time should be apparent from the following example:

MM = 60



This note occurs on the second half of the seventh beat or in absolute time 00:00:06:15

MM = 120



The note still occurs on the second half of the seventh beat but in absolute time it is now at approximately 00:00:03:07.

The Quarter Frame message is actually one part of the **MIDI Time Code** extension adopted by the MMA as a method of communicating SMPTE time through MIDI. **MTC's** main purpose is to allow accurate synchronization of MIDI events with other events on video and audio tape or film. Many film and video producers now regularly use it to cue sound effects from a sampler.



Quarter Frame is a 2 byte message comprised of the quarter frame status byte followed by a single data byte whose MSN defines its function and whose LSN conveys the function's data.

**11110001**    **00010001**  
Quarter frame    Frame MSN    Frame 16

Of course, it is impossible to convey a full rendition of SMPTE time code in a single data byte with only 128 combinations. (24 hours X 60 minutes X 60 seconds X 30 frames requires 2,592,000 combinations.) Therefore, a group of 8 sequential quarter frame messages is used to convey a full rendition of SMPTE time as in the following example:

**11110001 00000000** = frame LSN  
**11110001 00010000** = frame MSN  
**11110001 00100000** = second LSN  
**11110001 00110000** = second MSN  
**11110001 01000000** = minute LSN  
**11110001 01010000** = minute MSN  
**11110001 01100000** = hour LSN  
**11110001 01110110** = hour MSN & type of SMPTE

The complete message above reads:

**00:00:00:00** (at 30fps/non drop frame)  
Hours    Minutes    Seconds    Frames    Type of SMPTE code

To interpret the number of hours, minutes, seconds, or frames encoded in the data bytes of the message group above, the receiving device combines the least significant four bits of the appropriate LSN byte with the least significant bit (for frames and hours) or two bits (for minutes and seconds) of the appropriate MSN byte. Here is another example:

<b>11110001</b>	<b>00000011</b>	= frame LSN	_____	
<b>11110001</b>	<b>00010001</b>	= frame MSN	—————▶	<b>1 0011 (19)</b>
<b>11110001</b>	<b>00101111</b>	= second LSN	_____	
<b>11110001</b>	<b>00110010</b>	= second MSN	—————▶	<b>10 1111 (47)</b>
<b>11110001</b>	<b>01000101</b>	= minute LSN	_____	
<b>11110001</b>	<b>01010001</b>	= minute MSN	—————▶	<b>00 0111 (07)</b>
<b>11110001</b>	<b>01100100</b>	= hour LSN	_____	
<b>11110001</b>	<b>01110110</b>	= hr MSN & SMPTE type	—————▶	<b>110 0100 (04)</b>

**04:07:47:19** (at 30fps/non drop frame)  
Hours      Minutes      Seconds      Frames      Type of SMPTE code

A total of six bits (LSN and 2 bits of MSN) are needed to encode minute and second information because those temporal units require 60 unique numbers. Frames and hours, which require only 30 and 24 unique numbers apiece can be encoded in only 5 total bits. Six bit and five bit encoding still leaves several bits of the combined LSN and MSN nibbles unused. For frames, seconds, and minutes, these unused bits are reserved for future expansion. For hours, two of them are assigned to encode the kind of SMPTE being used.

SMPTE time coding actually incorporates four different methods of dividing seconds into frames. These methods are based on the frame rates of motion picture film (24 fps), European PAL television (25 fps), American NTSC black & white television (30 fps), and American NTSC Color television (30 fps drop frame or an average of 29.97 fps). In MIDI time code the selection of method is coded in bits 2 and 3 of the hour MSN data byte according to the following standard:

<b>_00_</b>	<b>= 24 fps</b>
<b>_01_</b>	<b>= 25 fps</b>
<b>_10_</b>	<b>= 30 fps drop frame</b>
<b>_11_</b>	<b>= 30 fps</b>

## System Common - Song Position Pointer (11110010)

Song position pointer is a three byte message comprised of the song position pointer status byte followed by a data byte representing the LSB of a location in a song and a data byte representing the MSB of the location.

**11110010**    **00000000**    **01000000**  
Song Position Pointer    LSB = 0 MIDI beats    MSB = 8192 MIDI beats

The pointer is really just a counter that increments one number for every MIDI Beat from the beginning of a song. A **MIDI Beat** is defined as six MIDI Clocks. A **MIDI Clock** is a single byte system real time message that a sending device transmits regularly like a metronome at the rate of 24 per quarter note. A MIDI Beat therefore equals a sixteenth note.

Quarter note	= 24
Dotted eighth	= 18
Triplet quarter	= 16
Eighth note	= 12
Dotted 16th	= 9
Triplet eighth	= 8
Sixteenth note	= 6
Triplet sixteenth	= 4
32 nd note	= 3
Triplet 32nd	= 2

The combination of LSB and MSB allows MIDI to pinpoint any one of 16,384 positions in a song with a quarter of a beat (sixteenth note) accuracy.

## System Common - Song Select (11110011)

Song select is a two byte message comprised of the song select status byte followed by a data byte representing the number of the song or pattern desired.

**11110011**    **00000001**  
Song select    Song number 1

This can be an effective message in selecting patterns from a drum machine or in chaining multiple sequences together. Note that it only selects a song or pattern; it does not play it. To start the song or pattern playing requires the system real time start or continue message discussed later.

**System Common - undefined (11110100)**

This unused status byte allows for future expansion of the MIDI spec.

**System Common - undefined (11110101)**

This unused status byte allows for future expansion of the MIDI spec.

**System Common - Tune Request (11110110)**

Tune request is a single byte message that tells any analog synthesizers in the setup to retune their tone generators. Since newer synthesizers are now almost exclusively digital and don't require periodic retuning, this message has become superfluous.

**System Common - End of Exclusive (11110111)**

The **End of Exclusive (EOX)** status byte is not used as a message by itself but rather as a caboose on a long train of system exclusive data bytes. Its function is to tell the receiving unit that the system exclusive information is complete and that the next byte will be the start of a new message.

**System Real Time - MIDI Clock (11111000)**

**MIDI Clock** is a single byte message designed to tell a receiving sequencer to step ahead by 1/24th of a quarter note. It is useful for synchronizing two or more sequencers to the same metronomic pulse. Because of its time critical nature the MIDI Clock message takes priority over other messages on the MIDI bus. This means that it is sent and received at exact, regular intervals even if this requires that it be inserted in the middle of another message.

**System Real Time - Undefined (11111001)**

This unused status byte allows for future expansion of the MIDI spec.

**System Real Time - Start (111111010)**

**Start** is a single byte message designed to command a receiving sequencer to move to the beginning of a song. The next MIDI Clock message then begins playing the song.

**System Real Time - Continue (111111011)**

**Continue** is a single byte message designed to signal a paused sequencer that it is about to begin at the point it stopped. The next MIDI clock then begins stepping through the song at that point.

**System Real Time - Stop (111111100)**

**Stop** is a single byte message which forces all receiving sequencers to immediately stop and to store their beat positions.

**SystemReal Time - Undefined (11111101)**

This unused status byte allows for future expansion of the MIDI spec.

**System Real Time - Active Sensing (111111110)**

**Active Sensing** is an optional single byte message that was designed to be sent throughout a MIDI setup at intervals of less than 300 milliseconds to insure against unexpected bus failures. After sensing the first active sensing message any MIDI units that respond to it expect to receive at intervals of no more than 300 milliseconds. If they do not sense the message within that time, they shut down all oscillators immediately. The main benefit of this message is for avoiding stuck notes if someone trips over a MIDI cable and disconnects the system between a note on and a note off message.

**System Real Time - System Reset (111111111)**

**System Reset** is a single byte message that reinitializes all receiving units. This means that it immediately shuts off all voices, stops any moving sequences, forces the sequencers to the beginning position, switches the units to local on, switches to omni on, poly mode, and clears any status bytes out of running status. It is the ultimate panic button in MIDI and should not be used very often.

## A Summary of All MIDI Bytes and What They Mean

### Data Bytes (MSB = 0)

00000000 - 01111111 Refines/qualifies Status Byte commands

### Status Bytes (MSB = 1)

**Channel Messages:** (xxxx tells channel number)  
(Includes **Voice** messages and 8 **Mode** messages)

1000xxxx	Note Off		
1001xxxx	Note On		
1010xxxx	Poly Pressure		0 - 31 = Two Byte Continuous Controllers (MSB)
1011xxxx	Control Change		32 - 63 = Two Byte Continuous Controllers (LSB)
1100xxxx	Program Change		64 - 69 = Switch Controllers
1101xxxx	Channel Pressure		70 - 95 = One Byte Continuous Controllers
1110xxxx	Pitch Bend		96 - 101 = Registered & Non-Registered Parameters
			102 - 119 = undefined
			120 - 127 = Mode Controllers

### System Messages:

#### System Exclusive:

11110000	System Exclusive		0 - 124 = Manufacturers Messages
			125 = Universal Non-Commercial Messages
			126 = Universal Non-Real Time Messages
			127 = Universal Real Time Messages

#### System Common:

11110001	Quarter Frame
11110010	Song Position Pointer
11110011	Song Select
11110100	undefined
11110101	undefined
11110110	Tune Request
11110111	End of Exclusive

#### System Real Time:

11111000	MIDI Clock
11111001	undefined
11111010	Start
11111011	Continue
11111100	Stop
11111101	undefined
11111110	Active Sensing
11111111	System Reset