# The MIDI Protocol

*Where examples of MIDI messages are presented, values are in hexadecimal*

The MIDI protocol consists of 'messages' which are designed to allow synthesizers and sequencers to communicate 'what sound to play' information.

Let's start with an example to illustrate a few concepts. Suppose the musician presses the middle-C key on a keyboard. The keyboard would send a 'note-on' message like this:

```
        9n    kk    vv
example:
        90    3C    40
        -----------------> time
```

where:

*n*
: is the *MIDI Channel*
`range: 0..F`

*kk*
: is a *key* pressed
`range: 00..7F`

*vv*
: is a *velocity* with which the key was *pressed*
`range: 00..7F`

So in the above example, the musician has pressed key '3C' (middle-C), with a velocity which is half-way along the range available velocities.

The MIDI device which receives this message will begin to sound the note middle-C, with the appropriate envelope et al for the device's capabilities and settings.

Now the musician releases the key, and a 'note-off' message is sent:

```
        8n    kk    vv
example:
        80    3C    33
        -----------------> time
```

where:

*vv*
: is a *velocity* with which the key was *released*
`range: 00..7F`

In this case, the musician has released the middle-C key, a bit more slowly than it was pressed.

The receiving MIDI device will now either cease or decay, as appropriate, the note middle-C. (note 1)

Some other typical MIDI messages are:

- Program (instrument) change (2 bytes)
- Pitch bend (2-3 bytes)
- Control Change, i.e. pedal/footswitch (2-3 bytes)
- Timing clock (1 byte)

As you can see, with only a few exceptions, MIDI messages tend to be very short.

MIDI Messages are all one-way. There are no 'acknowledgment' messages sent from the receiver back to the transmitter. This is an important point for the transmitter, because there is no guarantee that a given message is supported by the receiver.

A MIDI device will handle each message as best it can, or ignore it.

For example:

> a MIDI device which does not recognize the 'velocity' value in a note-on message, would play all notes at the same volume, regardless of the velocity value.

# MIDI Channels

Notice how we have already introduced the concept of a **MIDI Channel**. There are 16 MIDI Channels, numbered 1 to 16, where the values *n*=`0..F` correspond to MIDI channels 1...16, respectively

All messages such as note-on and note-off are specific to a MIDI Channel. This gives us 16 *independent* control channels for controlling MIDI devices.

A single device may be capable of sending or receiving messages on more than one MIDI channel. For example:

- A keyboard typically sends messages on only one channel at a time.
- A sequencer may send messages on any or all channels
- A sound-module is usually capable of receiving messages on several different channels at once.

# Status Bytes and Data Bytes

Notice also how in our examples, the range of values for key (*kk*) and velocity (*vv*) is only 0...7f

This is because all MIDI messages have simple structure consisting of:

- A **Status Byte**
  `range: 0x80..FF`
- Zero or more **Data Bytes**
  `range: 0x00..7F`

with the exception of System Exclusive messages, which have a

- A Status Byte
- Zero or more Data Bytes
- an End-Of-Exclusive status byte.

Status Bytes always have their Most Significant Bit set to 1, while Data Bytes have their Most Significant Bit set to 0.

As a general rule, when a MIDI device receives a new status byte, it takes on the appropriate state, regardless of whether the previous message was completed or not.

The only exception to this rule is System Real-Time messages, which may be inserted into the middle of other messages, without affecting those messages.

This basic structure of status and data bytes makes the MIDI protocol reasonably insensitve to single-bit errors in the data-stream.

# Types of MIDI messages

MIDI messages which are specific to a MIDI Channel are referred to as **Channel Messages**. MIDI messages which affect the entire MIDI system (or at least entire MIDI device) are known as **System Messages**. System Messages are not associated with a MIDI channel.

Channel and System messages are further subdivided into the following classes:

**Channel Voice Messages**
: Messages which start, alter or stop a sound or sounds being played.

**Channel Mode Messages**
: Control messages which affect the entire channel, such as switching from polyphonic-mode to monophonic-mode, and all-notes-off

**System Real-Time Messages**
: These are used by sequencers to regulate and synchronize timing. They consist of a status-byte only; no data-bytes are used.

**System Common Messages**
: Includes messages such as 'song position pointer', 'song select', MIDI Time Code Quarter Frame, which are used by sequencers.

**System Exclusive Messages**
: Generally used for device-specific extensions to the MIDI protocol. Each manufacturer is free to define their own System Exclusive Messages. Sample Dump Standard messages, MIDI Time Control Full Message, and other extensions are also implemented as System Exclusive messages

In general, System Exclusive Messages are the only MIDI messages which are more than a few bytes long.

# Running Status

Since MIDI messages are sent and interpreted in real-time, quite some emphasis is placed on reducing the volume of data which must be sent.

For ordinary note-on and note-off messages, it is quite common for several notes to be turned on or off, more or less at the same time.

Consider the following example:
A musician presses a chord on a MIDI keyboard consisting of 3 notes, say a C-chord. Lets suppose that the keyboard is set to MIDI Channel 3. The following messages would result:

```
  92    3C   44     92    40   40     92    43   3E
-------------------------------------------------> time
status  kk   vv   status  kk   vv   status  kk   vv
on,ch3  'C'       on,ch3  'E'       on,ch3  'G'
```

Notice how all the messages have exactly the same status-byte. Hence the status-byte is technically redundant for all but the 1st note-on message.

**Running Status** allows the subsequent MIDI messages to be sent without a status-byte. Only the data-bytes are sent.

In our example, the same messages could be sent as follows, using Running Status:

```
  92    3C   44   40   40   43   3E
-------------------------------------------------> time
status  kk   vv   kk   vv   kk   vv
on,ch3  'C'       'E'       'G'
```

Running Status:

- can only be used as long as the same status-byte still applies
  (goes without saying, really)
- is in force until a new status-byte is received
  (ie there is no timeout defined for Running Status)
- applies to **Channel Voice** messages and **Channel Mode** messages *only*
- is *not affected* by **System Real Time** messages
- is cancelled by:
  - Power-on
  - **System Exclusive** messages
  - **System Common** messages
- if the Running Status has been cancelled, subsequent data-bytes are ignored until a new status-byte is received

# Velocity=0 as Note-off

In order to extend the effectiveness of the Running Status concept, *by definition*, a note-on message of velocity=0 is interpreted as a note-off message of velocity=0x40.

Consider the following example:
The note middle-C is pressed, then released. Normally, this would result in something like the following messages:

```
  92    3C    44            82     3C    40
-----------------  ~  ----------------------> time
status  kk    vv          status   kk    vv
on,ch3  'C'               off,ch3  'C'
```

The same messages could also be sent as follows:

```
  92    3C    44            3C    00
-----------------  ~  ----------------------> time
status  kk    vv            kk    00
on,ch3  'C'                 'C'
```

Note that this time, we have paid a price for making the message shorter. We have lost the ability to include a 'release velocity'.

We should bear in mind that not all MIDI devices are capable of generating a meaningful 'release velocity'. Such a device is likely to send a note-on/velocity=0 message instead of a note-off message, since it is required to send a release velocity of 0x40 in any case (note 2),

Also consider that it is somewhat uncommon for the release velocity to have any effect on the sound being generated. Most timbres (voices) which imitate 'real' instruments would be unaffected by the value of the release velocity.

# Unsupported MIDI Messages

A MIDI device is not required to implement *all* possible features described in the MIDI specification.

The MIDI specification *does* require that all MIDI devices be able to receive all possible MIDI messages without causing internal errors. (note 3)

If a MIDI device receives a message which corresponds to a feature which has not been implemented on this device, it is *required* to simply ignore this message.

This requirement extends to the correct operation of Running Status, which may be affected (or not) by messages which have not been implemented.

For example:

a device which does not recognize the Active Sensing message should at least recognize that this message does *not* cancel the current Running Status.

This is one of the strengths of the MIDI standard, as it allows simple and inexpensive MIDI devices to co-exist and work with more sophisticated MIDI devices.

Note 1

Just because a device has received a note-off message does not automatically imply that the note should cease abruptly. Some sounds, such as organ and trumpet sounds will do so. Others, such as piano and guitar sounds, will decay (fade-out) instead, albeit more quickly after the note-off message is received.

Note 2

A device which does not implement 'note-on velocity' must send a note-on velocity of 0x40.
Similarly, a device which does not implement 'release velocity' must send a release velocity of 0x40. Such a device may use either a note-off/velocity=0x40 message, or a note-on/velocity=0 message, though the latter is preferred, as it allows Running Status to be used.

Note 3

Some MIDI devices may only have a MIDI-OUT port, in which case the requirement to receive all possible messages doesn't apply.

physical_layer.html Contents midi_modes.html