MIDI Specifications

MIDI is a standardized way for the communication between musical devices. It specifies the physical interface as well as the transmission protocol. There are different types of MIDI messages, for a full list please follow the links below.

Here a short overview: each message begins with a status byte, where the 8th (leftmost) bit is set. Thereafter additional data bytes are sent, where the leftmost bit is always zero. Accordingly, data bytes are always 7bit values (0-127 decimal, 0x00-0x7F hexadecimal), and status bytes are always between the range of 128-255 decimal, 0x80-0xFF hexadecimal. Sending data bytes with values >= 128 will violate the MIDI protocol and can lead to random behaviour at the receiver side, because the receiver could assume, that a new MIDI message begins (8th bit set → status byte).

The number of bytes which have to follow is strictly tied to the type of message, only exception are System Exclusive messages, where the data bytes are framed by a start and stop identifier.

Some MIDI messages like RPN, NRPN and Pitch Bender allow to combine two 7bit values to a 14bit value. This extends the value range to 0-16383 (decimal) or 0x0000-0x3FFF hexadecimal - more than enough for todays synthesizers (most of them still only support 7bit resolution).

MIDI bytes are mostly displayed in hexadecimal form to simplify the reading. Therefore we will continue with this value format below, and won't mention the appr. decimal values anymore.

Status bytes in the range of 0x80..0xEF contain a channel information, which allow to address up to 16 different listeners, or multiple listeners which are listen at the same time when they are listining the same channel. The MIDI channel is located within the first 4 bits of the status byte.

For Example:

```
0x90 identifies a Note On event at the first Channel
0x91 identifies a Note On event at the second Channel
...
0x9F identifies a Note On event at the 16th Channel
```

Thanks to this coding, it's very simple to identify the Channel, just read the rightmost hexadecimal digit.

The last 4 bits (leftmost hexadecimal digit) identifies the MIDI event type:

```
8: Note Off
9: Note On
A: Polyphonic Key Pressure (Poly Aftertouch)
B: Control Change (CC)
C: Program Change
D: Channel Pressure (Aftertouch)
E: Pitch Bender
```

Follow this link for a more detailed table, which also contains the number and purpose of the data bytes

F identifies a System Message which either addresses all listeners (there is no MIDI channel), or which addresses dedicated MIDI devices which are parsing for a SysEx (System Exclusive) stream.

E.g., MIOS “feels addressed” on SysEx messages which are starting with 0xF0 0x00 0x00 0x7E 0x40, followed by data byte which selects the device (Device ID), followed by a command to the operating system (e.g. Code Upload). SysEx messages should be finished with 0xF7, thereafter any other MIDI event can be sent again.

The last Status Bytes within the range of 0xF8..0xFF - also called “Realtime messages” - have (again) a special purpose, because it's allowed to send them at any time, even in between any other MIDI message without violating the protocol.

A typical realtime message is the MIDI clock 0xF8 - since it is allowed to send the clock at any time, regardless of the currently sent stream - the achievable latency is very low. In addition, realtime messages don't change the running status - more about this topic (status byte can be omitted if the same one was sent before) can be read in the MIDI spec.

Programming Examples

At the end some practical informations for MIOS programmers (examples are written in C):

MIDI bytes are sent with the MIOS_MIDI_TxBufferPut() function:

```
  MIOS_MIDI_TxBufferPut(0x90); // Note On, Channel #1 (we are counting from
1)
  MIOS_MIDI_TxBufferPut(0x40); // note number
  MIOS_MIDI_TxBufferPut(0x64); // velocity
```

will send a Note On event at Channel 0 (first channel, in most MIDI applications called Channel #1 when they are counting from 1). The Note number is 0x40 (E-3), the velocity is 0x64

The appr Note Off event starts either with 0x80, or with 0x90 (running status optimisation) and velocity 0x00:

```
  MIOS_MIDI_TxBufferPut(0x90); // Note On, Channel #1 (we are counting from
1)
  MIOS_MIDI_TxBufferPut(0x40); // note number
  MIOS_MIDI_TxBufferPut(0x00); // velocity == 0 -> same like Note Off
```

Other typical MIDI events are Controllers (CC), they begin with 0xbn (n = Channel):

```
  MIOS_MIDI_TxBufferPut(0xb8); // Note On, Channel #9 (we are counting from
1)
  MIOS_MIDI_TxBufferPut(0x01); // CC number #1 -> Modulation Wheel
  MIOS_MIDI_TxBufferPut(0x7f); // CC value
```

If the MIDIbox Link mechanism should be used to tunnel MIDI events through the MIDIbox Link Endpoint, a MIDI stream has to be framed by a begin and end function in the following way:

```
  MIOS_MIDI_BeginStream();
  MIOS_MIDI_TxBufferPut(0xcf); // Program Change, Channel #16 (we are
counting from 1)
  MIOS_MIDI_TxBufferPut(0x02); // Program Number
  MIOS_MIDI_EndStream();
```

And the last example shows, how to save MIDI bandwidth (the transmission of a single byte takes 320 uS) by using the Running Status feature (as mentioned above: so long the following status bytes are equal, we can omitt them):

```
  MIOS_MIDI_BeginStream();
  MIOS_MIDI_TxBufferPut(0x90); // Note On, Channel #1 (we are counting from
1)
  MIOS_MIDI_TxBufferPut(0x3c); // Note number for C-3
  MIOS_MIDI_TxBufferPut(0x7f); // velocity
  MIOS_MIDI_TxBufferPut(0x40); // Note number for E-3
  MIOS_MIDI_TxBufferPut(0x60); // velocity
  MIOS_MIDI_TxBufferPut(0x43); // Note number for G-3
  MIOS_MIDI_TxBufferPut(0x40); // velocity
  MIOS_MIDI_EndStream();
```

14-bit MIDI Messages

There are two ways to use 14-bit MIDI Messages; the trick is, to combine two 7-bit Messages to one 14-bit:

```
unsigned int MSB  = 127 << 7; // 16256
unsigned char LSB = 127; // 127
unsigned int largeNumber = MSB + LSB; // 16383:
```

- using RPNs or NRPNs
- sending two Controller Messages, eg:

```
CC 12, Effect Ctrl 1 (MSB = Most Significant Byte)
CC 44, Effect Ctrl 1 (LSB = Least Significant Byte)
```

Sending 14bit from one pot is only possible if you're hacking the code. Because Pots are being read as 10-bit value, you have to interpolate to 14 bit and implement a NRPN or dual-CC method.

Further informations:
Jglatt's Technical MIDI Specs Page or
Table 3: Summary of Control Change Messages (Data Bytes) for further examples and explanations

Tools & Helpers

ACMidiDefines – a definition listing to access Midi Events by Name

More Informations

More informations can be found online:

- MIDI Technical Fanatic's Brainwashing Center must-read!
- Jglatt's Technical MIDI Specs Page
- MIDI Introduction & Tutorial from midi.org
- Official MIDI Specifications Index from the MIDI Manufacturer's Association
  - Table 1: MIDI 1.0 Specification Message Summary
  - Table 2: Expanded Messages List (Status Bytes)
  - Table 3: Summary of Control Change Messages (Data Bytes)
- Excellent Article Summary and many Links at answers.com
- ASCII Conversion Table

From:
https://www.midibox.org/dokuwiki/ - **MIDIbox**

Permanent link:
https://www.midibox.org/dokuwiki/doku.php?id=midi_specification&rev=1162739105

Last update: **2006/11/05 15:06**