145-MIDI Parsing

MIDI parsing

Parsing MIDI in NetLinx code has a number of issues that the programmer (and system designer) need to be aware of. If you are totally unfamiliar with MIDI protocol, you should also take a look at TN 277 - MIDI Programming 101, and check out the MIDI message tables at the MMA website www.midi.org.

MIDI is faster than AXlink

The MIDI asynchronous data stream runs at 31250 baud, the AXlink data rate is 20.8K.
Data from a MIDI input to the master must be buffered at the AXlink device, or you will likely lose data.

The AXB-MIDI is better able to buffer the MIDI data stream than the older AXC-MIDI. The AXB-MIDI has a 3072 byte input buffer; the AXC-MIDI can only buffer 100 bytes. If you're trying to parse a lot of MIDI data you can get into trouble pretty quickly with only a 100 byte buffer.

With 3072 bytes buffered, the AXB-MIDI will be less likely to lose any data. The MIDI data stream can still get ahead of the AXB-MIDI, but as long as it's not a constant stream the AXB-MIDI will be able to catch up eventually.

The RS232 ports of the NI series of controllers can be set to 31250 baud, however wiring a MIDI device to an NI is outside the scope of this Tech Note.

MIDI protocol overview

MIDI bytes with value of $80 or higher are called *status* bytes, and indicate the type of message.
Status bytes are always denoted in hexadecimal.
Data bytes are always less than $80 (128 in decimal). Data bytes may be denoted in either decimal or hexadecimal.

Status bytes are further divided into *channel* messages ($80-$EF) and *system* messages ($F0-$FF).

Channel Messages

The upper nybble of a channel message status byte denotes the message type, the lower nybble corresponds to the MIDI channel. Low nybble $0 = MIDI channel 1, Low nybble $1 = MIDI channel 2, etc.

Most of the channel messages are *channel voice* messages, except for the controller messages using controllers 120127. Those are known as the *channel mode* messages.

Channel messages that may be significant for control systems are Note Off $8n, Note On $9n, Controller $Bn, and Program $Cn. (n = $0-$F corresponding to MIDI channel 1-16).
Note Off and Note On are sometimes used for preset selection.
Program is a MIDI device preset, typically.
Notable Controllers used are 0 (Program Bank MSB) & 32 (Program Bank LSB), 7 (Channel Volume MSB), others as needed.
Channel messages always have one or two data bytes, depending on the message. At first glance they appear simple to parse, however:

- When successive messages of the same type are sent, the data may be sent with no intervening status byte. This is known as *running status*. Effective MIDI parsing must be able to deal with running status by properly buffering the status byte. Devices that use running status will typically also send a Note On with velocity 0 instead of a Note Off in order to keep from having to change the status byte.
- There is no length byte or end byte, so there is no specific way to tell when the data ends – other than receiving another status byte.
- Due to quirks of the AXB-MIDI, running status, and the 64 byte AXlink limit, MIDI channel messages are frequently broken up and received in multiple events.

These channel message parsing issues were the main impetus for creating the NetLinx code attached to this tech note.

System Messages

System messages are divided into *system common* ($F0-$F7) and *system realtime* ($F8-$FF) messages.
Per the MIDI v1.0 spec, the system realtime messages should not affect the running status buffer, while the system common messages do.

Probably the most notable system message is the *system exclusive*, or *sysex*, which starts with a status byte of $F0, has 4 or more data bytes, and ends with a status byte of $F7. Sysex messages include manufacturer specific commands, and also Universal System Exclusives such as MIDI Machine Control and MIDI Show Control.

The AXB-MIDI handles the framing of MIDI system exclusive messages well, at least short ones < 64 bytes. You will typically get the whole message from $F0 to $F7 in one event. If they are longer than 64 bytes, they will always be broken up as 64 bytes is the longest string that can be sent across AXLink.
If all you are doing is parsing short system exclusive messages, you can probably do that without using the attached code.

Sysex messages are usually well-documented in their respective manuals. Just be sure to watch out for the data representation switching between decimal and hexadecimal depending on context.

MIDI Parsing Template

Attached to this tech note is a NetLinx workspace containing two small NetLinx programs that may be used as templates for MIDI parsing. One uses a module to pre-parse the data, the other does not. The code should be commented enough for a programmer well versed in parsing RS232 strings to use without too much trouble.