System Exclusive Messages

Goffredo Haus, editor

Fall 1995

System Exclusive	F0H

System messages are not assigned to any particular MIDI channel. They are recognized by MIDI receivers regardless of the basic channel to which they are set. System Exclusive messages, however, are an exception. Each instrument's System Exclusive messages (hereafter abbreviated as "Exclusive" messages) have their own special format according to an assigned manufacturer's ID number. Exclusive messages are used to send data such as patch parameters, sampler data, or a sequencer memory bulk dump. A format which is appropriate to the particular type of transmitter and receiver is required. For example, an Exclusive message which sets the feedback level for an operator in an FM digital synthesizer will have no corresponding or meaningful function in an analog synthesizer.

Since the purpose of MIDI is to connect many kinds of musical instruments and peripheral equipment, it is best not to use Exclusive messages to convey real-time performance information. Performance information is best sent via voice messages in real time. Receivers should ignore Exclusive messages with ID numbers that do not correspond to their own ID.

To avoid conflicts with non-compatible Exclusive messages, a specific ID number is granted to manufacturers of MIDI instruments by the MMA or JMSC. By agreement between the MMA and JMSC when an ID number is given, the Exclusive format which is used under that ID number must be published within one year. "Published", in this context, means not only utilizing the format, but also printing the information in the product's owner's manual and/or technical materials published by the manufacturer. This is consistent with one of the fundamental purposes of MIDI, which is to publicize information and foster compatibility.

Any manufacturer of MIDI hardware or software may use the system exclusive codes of any existing product without the permission of the original manufacturer. They may not modify or extend it in any way that conflicts with the original specification published by the designer. Once published, an Exclusive format is treated like any other part of the instruments MIDI implementation, so long as the new instrument remains within the definitions of the published specification.

Once an Exclusive format has been published, it should not be changed with the exception of bug fixes. If a new System Exclusive format is released, it should be published in the same manner as the first version.

- Universal System Exclusive Codes
- Sample Dump Standard
- ACK
- NAK
- CANCEL
- Wait
- Dump Header
- Data Packet
- Multiple Loop Point Messages
- Inquiry Message
- MIDI Time Code
- Quarter Frame Messages
- Quarter Frame Messages Implementation
- Full Message
- User Bits
- MIDI Cueing
- Description of MTC Cueing Set-Up Types
- Potential Problems
- MTC Signal Path Summary
- Reference

Three System Exclusive ID numbers have been set aside for specific purposes. 7DH is reserved for non-commercial use (e.g. schools, research, etc.) and is not to be used on any product leased to the public. 7EH (Non-Real Time) and 7FH (Real Time) are used for extensions to the MIDI specification. The standardized format for both Real Time and Non-Real

Time messages is as follows:

F0H
<ID number>
<channel number>
<sub-ID#1>
<sub-ID#2>
. . .
F7H

Since Non-Commercial codes would not be seen or used by an ordinary user, there is no need for a standard format.

A standard has been developed for sampler data dumps. It has been designed to work as an open or closed loop system. The closed loop system implements handshaking to improve speed and error recovery. This also accommodates machines that may need more time to process incoming data. The open loop system may be desired by those wishing to implement a simplified version with no handshaking.
The basic messages are Dump Request, ACK, NAK, Wait, Cancel, Dump Header, and Data Packets. These messages are described in detail below. The data formats are given in hexadecimal.

{Dump Request}
F0 7E cc 03 ss ss F7

with

cc: channel number
ss ss: requested sample, LSB first

Upon receiving this message, the sampler should check to see if the requested sample number falls in a legal range. If it is, the requested sample becomes the current sound number and is dumped to the requesting master following the procedure outlined below. If it is not within a legal range, the message should be ignored.

F0 7E cc 7F pp F7

with

cc: channel number
pp: packet number

This is the first handshaking flag. It means "Last data packet was received correctly. Start sending the next one". The packet number represents the packet being acknowledged as correct.

F0 7E cc 7E pp F7

with

cc: channel number
pp: packet number

This is the second handshaking flag. It means "Last data packet was received incorrectly. Please re-send". The packet number represents the packet being rejected.

F0 7E cc 7D pp F7

with

cc: channel number
pp: packet number

This is the third handshaking flag. It means "Abort dump". The packet number represents the packet on which the abort takes place.

F0 7E cc 7C pp F7

with

cc: channel number
pp: packet number

This is the fourth handshaking flag. It means "Do not send any more packets until told to do otherwise". This is important for systems in which the receiver (such as a computer) may need to perform other operations (such as disk access) before receiving the remainder of the dump. An ACK will continue the dump while a Cancel will abort the dump.

F0 7E cc 01 ss ss ee ff ff ff gg gg gg hh hh hh ii ii ii jj F7

with

cc: channel number
ss ss: sample number (LSB first)
ee: sample format (# of significant bits from 8-28)
ff ff ff: sample period (1/sample rate) in nanoseconds (LSB first)
gg gg gg: sample length in words (LSB first)
hh hh hh: sustain loop start point word number (LSB first)
ii ii ii: sustain loop end point word number (LSB first)
jj: loop type (00 = forward only, 01 = bàckward/forward)
7F: loop off

F0 7E cc 02 kk <120 bytes> ll F7

with

cc: channel number
kk: running packet count (0-127)
ll: checksum, XOR of (7E,cc,02,kk,<120 bytes>)

The total size of a data packet is 127 bytes. This is to prevent MIDI input buffer overflow in machines that may want to receive an entire message before processing it. 128 bytes, or 1/2 page of memory, is considered the smallest reasonable buffer for modern MIDI instruments.

Once a dump has been requested either from the front panel or over MIDI, the dump header is sent. After sending the header, the master must time out for at least two seconds, allowing the receiver to decide if it will accept the dump (enough memory, etc.). If the master receives a Cancel, it should abort the dump immediately. If it receives an ACK, it will start sending data packets. If it receives a Wait, it will pause indefinitely until another message is received. If nothing is received within the timeout, the master will assume an open loop and begin sending packets.

A data packet consists of its own header, a packet number, 120 data bytes, a checksum, and an End Of Exclusive (EOX). The packet number starts at 00 and increments with each new packet, resetting to 00 after it reaches 7FH. This is used by the receiver to distinguish between a new data packet and one being resent. This number is followed by 120 bytes of data which form 60 words (MSB first).

Each data byte consists of 7 bits. If the sample format is 8-14 bit, two bytes form a word. Sample formats of 15-21 bits require three bytes/word (yielding 40 words/packet). Sample formats of 22-28 bits require four bytes/word (yielding 30 words/packet). Information is left justified within the 7-bit bytes and unused bits are filled in with zeros. For example, the sample word FFFH would be sent as 01111111B 01111100B. The word FFFH happens to represent a full positive value (000H represents full negative). The checksum is the XOR of

(7E,<channel>,02,<packet number>,<120 bytes>)

When a sampler is receiving a data dump, it should keep a running checksum during reception. If the checksums match, it sends an ACK and wait for the next packet. If the checksums do not match, it sends a NAK and waits for the next packet. If the next packet number does not match the previous one and the sampler has no facility for receiving packets out of sequence, it should ignore the error and continue as if the checksum had matched.

When a sampler is sending a data dump, it should send a packet and watch its MIDI In port. If an ACK is received, it sends the next packet. If a NAK is received and the packet number matches that of the previous packet, it re-sends that packet. If the packet numbers do not match and the sampler has no facility to send packets out of sequence, it should ignore the NAK If a Wait is received, the sampler should watch its MIDI IN port indefinitely for another message and process it like a normal ACK, NAK, Cancel, or illegal message (which would usually abort the dump). If nothing is received in a specified time, the sampler can assume an open loop and send the next packet.

The packet numbers are included in the handshaking flags (ACK, NAK, Cancel, Wait) in order to accommodate future machines that might have the intelligence to re-transmit specific packets after the entire dump is completed or synchronization is lost.

This process continues until there are less than 121 bytes to send. The final data packet will still consist of 120 data bytes regardless of how many significant bytes actually remain. The unused bytes will be filled out with zeros. The receiver should receive and handshake on the last packet. If the receiver's memory becomes full, it should send a Cancel to the master.

These messages were added as an extension to the Sample Dump Standard. It allows for the definition of up to 16383 pairs of loop points per sample. This cures the shortcoming of the Sample Dump Standard allowing only 1 pair of loop points to be defined per sample. It also allows modification of loop points without also having to send the sample itself. These and any future extensions to the Sample Dump Standard will follow the Sub-ID #1 (05) of the Universal System Exclusive Real-Time message. The formats of these messages are as follows:

> {Loop Point Transmission (17 bytes)}
> F0 7E <channel> 05 01 ss ss bb bb cc dd dd dd ee ee ee F7

with

> F0 7E <channel>: Universal System Exclusive Non-Real Time header
> 05: Additional Loop information (sub-ID #1)

01: Multiple Loop messages (sub-ID #2)
ss ss: Sample Number (LSB first)
bb bb: Loop number (LSB first; 7F 7F = delete all loops)
cc: Loop type
00 = Forwards Only (unidirectional)
01 = Backwards/Forwards (bi-directional)
7F = Off
dd dd dd: Loop start address (in samples; LSB first)
ee ee ee: Loop end address (in samples; LSB first)
F7: EOX
{Loop Points Request (10 bytes)}
F0 7E <chan> 05 02 ss ss bb bb F7

with

F0 7E <chan>: Universal System Exclusive Non-Real Time Header
05: Additional Loop information (sub-ID #1)
02: Loop Points Request (sub-ID #2)
ss ss: Sample Number (LSB first)
bb bb: Loop Number (LSB first; 7F 7F = request all loops)
F7: EOX

One message is sent and one loop affected per loop request or transmission, with the obvious exceptions of 'Delete All Loops' and 'Request All Loops'. If a Loop Message is sent with the same number as an existing loop, the new information replaces the old. Loop number 00 00 is the same as the sustain loop defined in the Sample Dump Standard.

This message is sent by one device to request the identity of the receiving device. The format of the message is as follows:

F0 7E <channel> 06 01 F7

with

F0 7E <channel>: Universal System Exclusive Non-Real time header
06: General Information (sub-ID #1)
01: Device inquiry message (sub-ID #2)
F7: EOX

A device which receives the above message would respond as follows (note that if the channel number = 7FH that the device should respond regardless of what MIDI channel it is on):

F0 7E <channel> 06 02 mm ff ff dd dd ss ss ss ss F7

with

F0 7E <channel>: Universal System Exclusive Non-Real time header
06: General Information (sub-ID #1)

02: Device id message (sub-ID #2)
mm: Manufacturers System Exclusive id code
ff ff: Device family code (14 bits, LSB first)
dd dd: Device family member code (14 bits, LSB first)
ss ss ss ss: Software revision level. Format device specific
F7: EOX

Note that if the manufacturers id code is 00H then the above message is extended by two bytes to handle the additional manufacturers id code.

For device synchronization, MIDI Time Code uses two basic types of messages, described as Quarter Frame and Full. There is also a third, optional message for encoding SMPTE user bits.

Quarter Frame messages are used only while the system is running. They are rather like the PPQN or MIDI clocks to which we are accustomed. But there are several important ways in which Quarter Frame messages differ from the other systems.
As their name implies, they have fine resolution. If we assume 30 frames per second, there will be 120 Quarter Frame messages per second. This corresponds to a maximum latency of 8.3 milliseconds (at 30 frames per second), with accuracy greater than this possible within the specific device (which may interpolate in between quarter

frames to "bit" resolution). Quarter Frame messages serve a dual purpose: besides providing the basic timing pulse for the system, each message contains a unique nibble (four bits) defining a digit of a specific field of the current SMPTE time.

Quarter frames messages should be thought of as groups of eight messages. One of these groups encodes the SMPTE time in hours, minutes, seconds, and frames. Since it takes eight quarter frames for a complete time code message, the complete SMPTE time is updated every two frames. Each quarter frame message contains two bytes. The first byte is F1, the Quarter Frame System Common byte. The second byte contains a nibble that represents the message number (0 through 7), and a nibble for one of the digits of a time field (hours, minutes, seconds or frames).

{Quarter Frame Messages (2 bytes)}
F1 <message>

with

F1: System Common status byte
<message>: 0nnn dddd

with

dddd: 4 bits of binary data for this Message Type
nnn: Message Type:

0 = Frame count LS nibble
1 = Frame count MS nibble
2 = Seconds count LS nibble
3 = Seconds count MS nibble
4 = Minutes count LS nibble
5 = Minutes count MS nibble
6 = Hours count LS nibble
7 = Hours count MS nibble and SMPTE Type

After both the MS nibble and the LS nibble of the above counts are assembled, their bit fields are assigned as follows:

FRAME COUNT: xxx yyyyy

with

xxx: Undefined and reserved for future use. Transmitter must set these bits to 0 and receiver should ignore!
yyyyy: Frame number (0-29)

SECONDS COUNT: xx yyyyyy

with

xx: Undefined and reserved for future use. Transmitter must set these bits to 0 and receiver should ignore!
yyyyyy: Seconds Count (0-59)

MINUTES COUNT: xx yyyyyy

with

xx: Undefined and reserved for future use. Transmitter must set these bits to 0 and receiver should ignore!
yyyyyy: Minutes Count (0-59)

HOURS COUNT: x yy zzzzz

with

x: Undefined and reserved for future use. Transmitter must set this bit to 0 and receiver should ignore!
yy: Time Code Type:
0 = 24 Frames/Second
1 = 25 Frames/Second
2 = 30 Frames/Second (Drop-Frame)
3 = 30 Frames/Second (Non-Drop)
zzzzz: Hours Count (0-23)

When time code is running in the forward direction, the device producing the MIDI Time Code will send Quarter Frame messages at quarter frame intervals in the following order:

F1 0X
F1 1X
F1 2X
F1 3X
F1 4X
F1 5X

F1 6X
F1 7X

after which the sequence repeats itself, at a rate of one complete 8-message sequence every 2 frames (8 quarter frames). When time code is running in reverse, the quarter frame messages are sent in reverse order, starting with F1 7X and ending with F1 0X. Again, at least 8 quarter frame messages must be sent. The arrival of the F1 0X and F1 4X messages always denote frame boundaries.

Since 8 quarter frame messages are required to definitely establish the actual SMPTE time, timing lock cannot be achieved until the reader has read a full sequence of 8 messages, from first message to last. This will take from 2 to 4 frames to do, depending on when the reader comes on line.

During fast forward, rewind or shuttle modes, the time code generator should stop sending quarter frame messages, and just send a Full Message once the final destination has been reached. The generator can then pause for any devices to shuttle to that point, and resume by sending quarter frame messages when play mode is resumed. Time is considered to be "running" upon receipt of the first quarter frame message after a Full Message.

Do not send quarter frame messages continuously in a shuttle mode at high speed,since this unnecessarily clogs the MIDI data lines. If you must periodically update a device's time code during a long shuttle, then send a Full Message every so often.

The quarter frame message F1 0X (Frame Count LS nibble) must be sent on a frame boundary. The frame number indicated by the frame count is the number of the frame which starts on that boundary. This follows the same convention as normal SMPTE longitudinal time code, where bit 00 of the 80-bit message arrives at the precise time that the frame it represents is actually starting. The SMPTE time will be incremented by 2 frames for each 8-message sequence, since an 8-message sequence will take 2 frames to send.

Another way to look at it is: when the last quarter frame message (F1 7X) arrives and the time can be fully assembled, the information is now actually 2 frames old. A receiver of this time must keep an internal offset of +2 frames for displaying. This may seem unusual, but it is the way normal SMPTE is received and also makes backing up (running time code backwards) less confusing - when receiving the 8 quarter frame messages backwards, the F1 0X message still falls on the boundary of the frame it represents.

Each quarter frame message number (0->7) indicates which of the 8 quarter frames of the 2-frame sequence we are on. For example, message 0 (F1 0X) indicates quarter frame 1 of frame #1 in the sequence, and message 4 (F1 4X) indicates quarter frame 1 of frame #2 in the sequence. If a reader receives these message numbers in descending sequence, then it knows that time code is being sent in the reverse direction. Also, a reader can come on line at any time and know exactly where it is in relation to the 2-frame sequence, down to a quarter frame accuracy.

It is the responsibility of the time code reader to insure that MTC is being properly interpreted. This requires waiting a sufficient amount of time in order to achieve time code lock, and maintaining that lock until synchronization is dropped. Although each passing quarter frame message could be interpreted as a relative quarter frame count, the time code reader should always verify the actual complete time code after every 8-message sequence (2 frames) in order to guarantee a proper lock. If synchronization is dropped the transmitter should send a NAK message. The receiver should interpret this as "tape has stopped" and should turn of any lingering notes, etc.

For example, let's assume the time is 01:37:52:16 (30 frames per second, non-drop). Since the time is sent from least to most significant digit, the first two Quarter Frame messages will contain the data 16 (frames), the second two will contain the data 52 (seconds), the third two will represent 37 (minutes), and the final two encode the 1 (hours and SMPTE Type). The Quarter Frame Messages description defines how the binary data for each time field is spread across two nibbles. This scheme (as opposed to simple BCD) leaves some extra bits for encoding the SMPTE type (and for future

use).
Now, let's convert our example time of 01:37:52:16 into Quarter Frame format, putting in the correct hexadecimal conversions:

F1 00
F1 11
10H = 16 decimal
F1 24
F1 33
34H = 52 decimal
F1 45
F1 52
25H = 37 decimal
F1 61
F1 76
01H = 01 decimal (SMPTE Type is 30 frames/non-drop)

Note that the value transmitted is "6" because the SMPTE Type (11 binary) is encoded in bits 5 and 6.
For SMPTE Types of 24, 30 drop frame, and 30 non-drop frame, the frame number will always be even. For SMPTE Type of 25, the frame number may be even or odd, depending on which frame number the 8-message sequence had started. In this case, you can see where the MIDI Time Code frame number would alternate between even and odd every second.

MIDI Time Code will take a very small percentage of the MIDI bandwidth. The fastest SMPTE time rate is 30 frames per second. The specification is to send 4 messages per frame - in other words, a 2-byte message (640 microseconds) every 8.333 milliseconds. This takes 7.68 % of the MIDI bandwidth - a reasonably small amount. Also, in the typical MIDI Time Code systems we have imagined, it would be rare that normal MIDI and MIDI Time Code would share the same MIDI bus at the same time.
Note that, when a VITC signal drives a MIDI Time Code system through a SMPTE-to-MIDI converter, the MIDI Time Code frames do not advance by two as expected. They may advance by one or not at all, and time code which was even frame numbers may become odd frame numbers. To accomplish synchronization, it necessary to wait until the first four (at least) bits of the SMPTE have been received before sending the MTC so you know if the frame has advanced or not.

Quarter Frame messages handle the basic running work of the system. But they are not suitable for use when equipment needs to be fast-forwarded or rewound, located or cued to a specific time, as sending them continuously at accelerated speeds would unnecessarily clog up or outrun the MIDI data lines. For these cases, Full Messages are used, which encode the complete time into a single message. After sending a Full Message, the time code generator can pause for any mechanical devices to shuttle (or "autolocate") to that point, and then resume running by sending quarter frame messages.

> {Full Message - (10 bytes)}
> F0 7F <chan> 01 <sub-ID 2> hr mn sc fr F7

with

> F0 7F: Real Time Universal System Exclusive Header
> <chan>: 7F (message intended for entire system)
> 01: <sub-ID 1>, 'MIDI Time Code'
> <sub-ID 2>: 01, Full Time Code Message
> hr: hours and type: 0 yy zzzzz
>> with yy: type
>>> 00 = 24 Frames/Second
>>> 01 = 25 Frames/Second
>>> 10 = 30 Frames/Second (drop frame)
>>> 11 = 30 Frames/Second (non-drop frame)
>>
>> zzzzz: Hours (00->23)
>
> mn: Minutes (00->59)
> sc: Seconds (00->59)
> fr: Frames (00->29)
> F7: EOX

Time is considered to be "running" upon receipt of the first Quarter Frame message after a Full Message.

"User Bits" are 32 bits provided by SMPTE for special functions which vary with the application, and which can be programmed only from equipment especially designed for this purpose. Up to four characters or eight digits can be written. Examples of use are adding a date code or reel number to the tape. The User Bits tend not to change throughout a run of time code.

{User Bits Message - (15 bytes)}
F0 7F <chan> 01 <sub-ID 2> u1 u2 u3 u4 u5 u6 u7 u8 u9 F7

with

F0 7F: Real Time Universal System Exclusive Header
<chan>: 7F (message intended for entire system)
01: <sub-ID 1>, MIDI Time Code
<sub-id 2>: 02, User Bits Message

- u1: 0000aaaa
- u2: 0000bbbb
- u3: 0000cccc
- u4: 0000dddd
- u5: 0000eeee
- u6: 0000ffff
- u7: 0000gggg
- u8: 0000hhhh
- u9: 000000ii
- F7: EOX

These nibble fields decode in an 8-bit format:

aaaabbbb ccccdddd eeeeffff gggghhhh ii

It forms 4 8-bit characters, and a 2 bit Format Code. u1 through u8 correspond to SMPTE Binary Groups 1 through 8. u9 are the two Binary Group Flag Bits, as defined by SMPTE. This message can be sent whenever the User Bits values must be transferred to any devices down the line. Note that the User Bits Message may be sent by the MIDI Time Code Converter at any time. It is not sensitive to any mode.

MIDI Cueing uses Set-Up Messages to address individual units in a system (a "unit" can be a multitrack tape deck, a VTR, a special effects generator, MIDI sequencer, etc).

Of 128 possible event types, 19 are currently defined.

{Set-Up Messages (13 bytes plus any additional information)}
F0 7E <chan> 04 <sub-ID 2> hr mn sc fr ff s1 sm <add. info.> F7

with

F0 7E: Non-Real Time Universal System Exclusive Header
<chan>: Channel number
04: <sub-ID 1>, MIDI Time Code

<sub-ID 2>: Set-Up Type

00 = Special
01 = Punch In points
02 = Punch Out points
03 = Delete Punch In point
04 = Delete Punch Out point
05 = Event Start points
06 = Event Stop points
07 = Event Start points with additional info
08 = Event Stop points with additional info
09 = Delete Event Start point
0A = Delete Event Stop point
0B = Cue points
0C = Cue points with additional info
0D = Delete Cue point
0E = Event Name in additional info

hr: hours and type: 0 yy zzzzz

with yy: type

00 = 24 Frames/Second
01 = 25 Frames/Second
10 = 30 Frames/Second (drop frame)
11 = 30 Frames/Second (non-drop frame)

zzzzz: Hours (00->23)
mn: Minutes (00->59)
sc: Seconds (00->59)
fr: Frames (00->29)
ff: Fractional Frames (00-99)

sl, sm: Event Number (LSB first)
<add. info.>
F7: EOX

- 00: Special refers to the set-up information that affects a unit globally (as opposed to individual tracks, sounds, programs, sequences, etc.). In this case, the Special Type takes the place of the Event Number. Five are defined. Note that types 01 00 through 04 00 ignore the event time field.
- 00 00: Time Code Offset refers to a relative Time Code offset for each unit. For example, a piece of video and a piece of music that are supposed to go together may be created at different times, and more than likely have different absolute time code positions - therefore, one must be offset from the other so that they will match up. Just like there is one master time code for an entire system, each unit only needs one offset value per unit.
- 01 00: Enable Event List means for a unit to enable execution of events in its list if the appropriate MTC or SMPTE time occurs.
- 02 00: Disable Event List means for a unit to disable execution of its event list but not to erase it. This facilitates an MTC Event Manager in muting particular devices in order to concentrate on others in a complex system where many events occur simultaneously.
- 03 00: Clear Event List means for a unit to erase its

entire event list.

- 04 00: System Stop refers to a time when the unit may shut down. This serves as a protection against Event Starts without matching Event Stops, tape machines running past the end of the reel, and so on.
- 05 00: Event List Request is sent by a master to an MTC peripheral. If the device ID (Channel Number) matches that of the peripheral, the peripheral responds by transmitting its entire cue list as a sequence of Set Up Messages, starting from the SMPTE time indicated in the Event List Request message.
- 01/02: Punch In and Punch Out refer to the enabling and disabling of record mode on a unit. The Event Number refers to the track to be recorded. Multiple punch in/punch out points (and any of the other event types below) may be specified by sending multiple Set-Up messages with different times.
- 03/04: Delete Punch In or Out deletes the matching point (time and event number) from the Cue List.
- 05/06: Event Start and Stop refer to the running or playback of an event, and imply that a large sequence of events or a continuous event is to be started or stopped. The event number refers to which event on the targeted slave is to be played. A single event (i.e. playback of a specific sample, a fader movement on an automated console, etc.) may occur several times throughout a given list of cues. These events will be

represented by the same event number, with different Start and Stop times.

- 07/08: Event Start and Stop with Additional Information refer to an event (as above) with additional parameters transmitted in the Set Up message between the Time and EOX. The additional parameters may take the form of an effects unit's internal parameters, the volume level of a sound effect, etc. See below for a description of additional information.
- 09/0A: Delete Event Start/Stop means to delete the matching (event number and time) event (with or without additional information) from the Cue List.
- 0B: Cue Point refers to individual event occurrences, such as marking "hit" points for sound effects, reference points for editing, and so on. Each Cue number may be assigned to a specific reaction, such as a specific one-shot sound event (as opposed to a continuous event, which is handled by Start/Stop). A single cue may occur several times throughout a given list of cues. These events will be represented by the same event number, with different Start and Stop times.
- 0C: Cue Point with Additional Information is exactly like Event Start/Stop with Additional Information, except that the event represents a Cue Point rather than a Start/Stop Point.
- 0D: Delete Cue Point means to Delete the matching (event number and time) Cue Event with or without

additional information from the Cue List.
- 0E: Event Name in Additional Information. This merely assigns a name to a given event number. It is for human logging purposes. See Additional Information description.

- EVENT TIME: This is the SMPTE/MIDI Time Code time at which the given event is supposed to occur. Actual time is in 1/l00th frame resolution, for those units capable of handling bits or some other form of sub-frame resolution, and should otherwise be self-explanatory.
- EVENT NUMBER: This is a fourteen-bit value, enabling 16384 of each of the above types to be individually addressed. "sl" is the 7 LS bits, and "sm" is the 7 MS bits.
- ADDITIONAL INFORMATION: Additional information consists of a nibblized MIDI data stream, LS nibble first. The exception is Set-Up Type 0E, where the additional information is nibblized ASCII, LS nibble first. An ASCII newline is accomplished by sending CR and LF in the
- ASCII. CR alone functions solely as a carriage return, and LF alone functions solely as a LineFeed.

For example, a MIDI Note On message such as 91 46 7F would be nibblized and sent as 01 09 06 04 0F 07. In this way, any device can decode any message regardless of who it was intended for. Device-specific messages should be sent as nibblized MIDI System Exclusive messages.

There is a possible problem with MIDI merger boxes improperly handling the F1 message, since they do not currently know how many bytes are following. However, in typical MIDI Time Code systems, we do not anticipate applications where the MIDI Time Code must be merged with other MIDI signals occurring at the same time.
Please note that there is plenty of room for additional set-up types, etc., to cover unanticipated situations and configurations.
It is recommended that each MTC peripheral power up with its MIDI Manufacturer's System Exclusive ID number as its default channel/device ID. Obviously, it would be preferable to allow the user to change this number from the device's front panel, so that several peripherals from the same manufacturer may have unique IDs within the same MTC system.

Data sent between the Master Time Code Source (which may be, for example, a Multitrack Tape Deck with a SMPTE Synchronizer) and the MIDI Time Code Converter is always SMPTE Time Code.
Data sent from the MIDI Time Code Converter to the Master Control/Cue Sheet (note that this may be a MTC-equipped tape deck or mixing console as well as a cue-sheet) is always MIDI Time Code. The specific MIDI Time Code

messages which are used depend on the current operating mode, as explained below.

PLAY MODE : The Master Time Code Source (tape deck) is in normal PLAY MODE at normal or vari-speed rates. The MIDI Time Code Converter is transmitting Quarter Frame (F1) messages to the Master Control/Cue Sheet. The frame messages are in ASCENDING order, starting with "F1 0X" and ending with "F1 7X". If the tape machine is capable of play mode in REVERSE, then the frame messages will be transmitted in REVERSE sequence starting with "F1 7X" and ending with "F1 0X".

CUE MODE: The Master Time Code Source is being "rocked", or "cued" by hand. The tape is still contacting the playback head so that the listener can cue, or preview the contents of the tape slowly. The MIDI Time Code Converter is transmitting FRAME (F1) messages to the Master Control/Cue Sheet. If the tape is being played in the FORWARD direction, the frame messages are sent in ASCENDING order, starting with "F1 0X" and ending with "F1 7X". If the tape machine is played in the REVERSE direction, then the frame messages will be transmitted in REVERSE sequence, starting with "F1 7X" and ending with "F1 0X". Because the tape is being moved by hand in Cue Mode, the tape direction can change quickly and often. The order of the Frame Message sequence must change along with the tape direction.

FAST FORWARD/REWIND MODE: In this mode, the tape is in

a high-speed wind or rewind, and is not touching the playback head. No "cueing" of the taped material is going on. Since this is a "search" mode, synchronization of the Master Control/Cue Sheet is not as important as in the Play or Cue Mode. Thus, in this mode, the MIDI Time Code Converter only needs to send a "Full Message" every so often to the Cue Sheet. This acts as a rough indicator of the Master's position. The SMPTE time indicated by the "Full Message" actually takes effect upon the reception of the next "F1" quarter frame message (when "Play Mode" has resumed).
SHUTTLE MODE: This is just another expression for "Fast-Forward/Rewind Mode".

- SMPTE 12M (ANSI V98.12M-1981).

[Index | Main Paragraph]