MIDI Time Code

http://academic.pgcc.edu/~njudy/mt/MIDI/mtc.html

20180101

Contents

FORE

This document is 'found' at the noted source and represented (and perhaps reformatted) here as it had been transcribed primarily for ease of access, portability and readability. Since the individual editing/transcribing this to markdown->pdf is a patent artist and/or illustrator, we have replaced gif image figures with vector images and may have presented tables or other formatting options where it was perhaps found appropriate.

This document is part of a personal collection of three documents:

1. MIDI Specification, Updated ~1996
2. MIDI Time Code
3. MIDI Specification (This Document)

While these documents are fun and easy to read, it would be advisable to direct you to https://www.midi.org/specifications-old/item/the-midi-1-0-specification for official concurrent reference including addendas and so forth.

MIDI Time Code

MIDI Time Code (MTC) is a sub-protocol within MIDI, and is used to keep 2 devices that control some sort of timed performance (ie, maybe a sequencer and a video deck) in sync. MTC messages are an alternative to using MIDI Clocks and Song Position Pointer messages. MTC is essentially SMPTE mutated for transmission over MIDI. SMPTE timing is referenced from an absolute "time of day". On the other hand, MIDI Clocks and Song Position Pointer are based upon musical beats

from the start of a song, played at a specific Tempo. For many (non-musical) cues, it's easier for humans to reference time in some absolute way rather than based upon musical beats at a certain tempo.

There are several MIDI messages which make up the MTC protocol. All but one are specially defined SysEx messages.

The most important message is the Quarter Frame message (which is not a SysEx message). It has a status of 0xF1, and one subsequent data byte. This message is sent periodically to keep track of the running SMPTE time. It's analogous to the MIDI Clock message. The Quarter Frame messages are sent at a rate of 4 per each SMPTE Frame. In other words, by the time that a slave has received 4 Quarter Frame messages, a SMPTE Frame has passed. So, the Quarter Frame messages provide a "sub-frame" clock reference. (With 30 fps SMPTE, this "clock tick" happens every 8.3 milliseconds).

But the Quarter Frame is more than just a quarter frame "clock tick". The Quarter Frame message's data byte contains the SMPTE time (ie, hours, minutes, seconds, and frames). SMPTE time is normally expressed in 80 bits. Obviously, this is too many bits to be contained in 1 8-bit data byte. So, each Quarter Frame message contains just one piece of the time (for example, one Quarter Frame may contain only the hours). In order to get the entire SMPTE time at any given point, a slave needs to receive several Quarter Frame messages, and piece the current SMPTE time together from those messages. It takes 8 Quarter Frame messages to convey the current SMPTE time. In other words, by the time that a slave can piece together the current SMPTE time, two SMPTE frames have passed (ie, since there are 4 Quarter Frame messages in each frame). So, MTC's version of SMPTE time actually counts in increments of 2 SMPTE Frames per each update of the current SMPTE time.

The first (of 8) Quarter Frame message contains the low nibble (ie, bits 0 to 3) of the Frame Time. The second Quarter Frame message contains the high nibble (ie, bits 4 to 7) of the Frame Time. The third and fourth messages contain the low and high nibbles of the Seconds Time. The fifth and sixth messages contain the low and high nibbles of the Minutes Time. The seventh and eighth messages contain the low and high nibbles of the Hours Time. The eighth message also contains the SMPTE frames-per-second Type (ie, 24, 25, 30 drop, or 30 fps). If you were to break up the Quarter Frame's data byte into its 7 bits, the format is:

```
0nnn dddd
```

where `nnn` is one of 7 possible values which tell you what `dddd` represents. Here are the 7 values, and what each causes `dddd` to represent.

Value	dddd
0	Current Frames Low Nibble
1	Current Frames High Nibble
2	Current Seconds Low Nibble
3	Current Seconds High Nibble
4	Current Minutes Low Nibble
5	Current Minutes High Nibble
6	Current Hours Low Nibble
7	Current Hours High Nibble and SMPTE Type

`0xF1 0x25`	means that the 5 is the low nibble of the Seconds Time (because nnn is 2). If the following Quarter Frame is subsequently received,
`0xF1 0x32`	then this means that 2 is the high nibble of the Seconds Time. Therefore, the current SMPTE Seconds is `0x25` (ie, 37 seconds).

In the data byte for the Hours High Nibble and SMPTE Type, the bits are interpreted as follows:

`0nnn x yy d`

where `nnn` is 7. `x` is unused and set to 0. `d` is bit 4 of the Hours Time. `yy` tells the SMPTE Type as follows:

yy	fps
0	24
1	25
2	30 (Drop-Frame)
3	30

When MTC is running in the forward direction (ie, time is advancing), the Quarter Frame messages are sent in the order of Frames Low Nibble to Hours High Nibble. In other words, the order looks something like this:

`0xF1 0x0n`	where `n` is the current Frames Low Nibble

0xF1 0x1n	where n is the current Frames High Nibble
0xF1 0x2n	where n etc.
0xF1 0x3n	
0xF1 0x4n	
0xF1 0x5n	
0xF1 0x6n	
0xF1 0x7n	When MTC is running in reverse (ie, time is going backwards), these are sent in the opposite order, ie, the Hours High Nibble is sent first and the Frames Low Nibble is last. The arrival of the 0xF1 0x0n and 0xF1 0x4n messages always denote where SMPTE Frames actually occur in realtime.

Since 8 Quarter Frame messages are required to piece together the current SMPTE time, timing lock can't be achieved until the slave has received all 8 messages. This will take from 2 to 4 SMPTE Frames, depending upon when the slave comes online.

The Frame number (contained in the first 2 Quarter Frame messages) is the SMPTE Frames Time for when the first Quarter Frame message is sent. But, because it takes 7 more Quarter Frames to piece together the current SMPTE Time, when the slave does finally piece the time together, it is actually 2 SMPTE Frames behind the real current time. So, for display purposes, the slave should always add 2 frames to the current time.

For cueing the slave to a particular start point, Quarter Frame messages are not used. Instead, an MTC Full Frame message should be sent. The Full Frame is a SysEx message that encodes the entire SMPTE time in one message as so (in hex):

```
F0 7F cc 01 01 hr mn sc fr F7
```

cc is the SysEx channel (0 to 127). It is suggested that a device default to using its Manufacturer's SysEx ID number for this channel, giving the musician the option of changing it. Channel number 0x7F is used to indicate that all devices on the daisy-chain should recognize this Full Frame message.

The `hr`, `mn`, `sc`, and `fr` are the hours, minutes, seconds, and frames of the current SMPTE time. The hours byte also contains the SMPTE Type as per the Quarter Frame's Hours High Nibble message.

The Full Frame simply cues a slave to a particular SMPTE time. The slave doesn't actually start running until it starts receiving Quarter Frame messages. (Which implies that a slave is stopped whenever it is not receiving Quarter Frame messages). The master should pause after sending a Full Frame, and before sending a Quarter Frame, in order to give the slave time to cue to the desired SMPTE time.

During fast forward or rewind (ie, shuttle) modes, the master should not continuously send Quarter Frame messages, but rather, send Full Frame messages at regular intervals.

SMPTE also provides for 32 “user bits”, information for special functions which vary with each product. (Usually, these bits can only be programmed from equipment that supports such). Upto 4 characters or 8 digits can be written. Examples of use are adding a date code or reel number to a tape. The user bits tend not to change throughout a run of time code, so rather than stuffing this information into a Quarter Frame, MTC provides a separate SysEx message to transmit this info.

```
F0 7F cc 01 02 u1 u2 u3 u4 u5 u6 u7 u8 u9 F7
```

`cc` is the SysEx channel (0 to 127). Only the low nibble of each of the first 8 data bytes is used. Only the 2 low bits of u9 is used.

```
u1 = 0000aaaa
u2 = 0000bbbb
u3 = 0000cccc
u4 = 0000dddd
u5 = 0000eeee
u6 = 0000ffff
u7 = 0000gggg
u8 = 0000hhhh
u9 = 000000ii
```

These nibbles decode into an 8-bit format of `aaaabbbb ccccdddd eeeeffff ggggghhhh ii`. It forms 4 8-bit characters, and a 2 bit Format Code. u1 through u8 correspond to the SMPTE Binary Groups 1 through 8. u9 are the 2 Binary Group Flag Bits, defined by SMPTE.

The Users Bits messages can be sent at any time, whenever these values must be passed to some device on the daisy-chain.

Notation Information

There are two Notation Information messages which can be used to setup a device that needs to interact with the musician using musical bars and beats.

The Time Signature message can setup Time Signature or indicate a change of meter.

`F0 7F cc 03 ts ln nn dd qq [nn dd...] F7`

`cc` is the SysEx channel (0 to 127).

`ts` is 02 if the Time Signature is to be changed now, or 42 if the Time Signature is to be changed at the end of the currently playing measure.

`ln` is the number of data bytes following this field. Normally, this will be a 3 if there is not a compound time signature in the measure.

`nn dd` are the Numerator and Denominator of the Time Signature, respectively. Like with MIDI File Format's Time Signature MetaEvent, the Denominator is expressed as a power of 2.

`qq` is the number of notated 32nd notes in a MIDI quarter note. Again, this is similiar to the same field in MIDI File Format's Time Signature MetaEvent.

`[nn dd …]` are optional, additional pairs of num/denom, to define a compound time signature within the same measure.

The Bar Marker message indicates the start of a musical measure. It could also be used to setup and mark off bars of an introductory "count down".

`F0 7F cc 03 01 lb mb F7`

`cc` is the SysEx channel (0 to 127).

`lb mb` is the desired bar number, with the LSB first (ie, Intel order). This is a signed 14-bit value (low 7 bits are in `lb`, right-justified, and bits 8 to 14 are in `mb`, right-justified). Zero and negative numbers up to -8,190 indicate count off measures. For example, a value of -1 (ie, `lb mb` = `7F 7F`) means that there is a one measure introduction. A value of zero would indicate no count off. Positive values indicate measures of the piece. The first measure is bar 1 (ie, `lb mb` = `01 00`). A maximum neg number (`lb mb` = `00 40`) indicates "stopped play" condition. A maximum positive value (`lb mb` = `7E 3F`) indicates running condition, but no idea about measure number. This would be used by a device wishing to mark the passage of measures without keeping track of the actual measure number.

Setup Message

The Setup message can be used to implement one of 19 defined "events". A master device uses this message to tell slave units what "events" to perform, and when to perform those events. Here's the general template for the message.

`F0 7F cc 04 id hr mn sc fr ff sl sm [more info] F7`

`cc` is the SysEx channel (0 to 127).

`hr mn sc fr ff` is the SMPTE time when the event is to occur. This is just like the Full Frame message, except that there is also a fractional frame parameter, `ff`, which is $\frac{1}{100}$ of a frame (ie, a value from 0 to 99).

`sl sm` is this event's 14-bit Event Number (0 to 16,383). `sl` is bits 0 to 6, and `sm` is bits 7 to 13.

`id` tells what this Event Type is. Depending upon the Type, the message may have additional bytes placed where is. The following values for Event Types are defined, and here's what each does.

Special (00)

Contains the setup information that affects a device globally, as opposed to individual tracks, sounds, programs, sequences, etc.). In this case, the Event Number is actually a word which further describes what the event is, as so:

Time Code Offset (00 00) 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 likely have different absolute time code positions. Therefore, one must be offset from the other so that they will match up. Each slave on the daisy-chain needs its own offset so that all can be matched up to the master's SMPTE start time.

Enable Event List (01 00) means for a slave to enable execution of events in its internal "list of events" when each one's respective SMPTE time occurs.

Disable Event List (02 00) means for a slave to disable execution of events in its internal "list of events", but not to erase the list.

Clear Event List (03 00) means for a slave to erase all events in its internal list.

System Stop (04 00) refers to a time when the slave may shut down. This serves as a protection against Event Starts without Event Stops, tape machines running past the end of a reel, etc.

Event List Request (05 00) is sent by the master, and requests the slave (whose channel matches the message) to send all events in its list as a series of Setup messages, starting from the SMPTE time in this message.

> NOTE: For the first 5 Special messages, the SMPTE time isn't used and is ignored.

Punch In (01) and Punch Out (02)

These refer to the enabling and disabling of record mode on a slave. The Event Number refers to the track to be recorded. Multiple Punch In and Punch Out points (and any of the other Event Types below) may be specified by sending multiple Setup messages with different SMPTE times.

Delete Punch In (03) and Delete Punch Out (04)

Deletes the Punch In or Punch Out (with the matching Event Number and SMPTE Time) from the slave's event list. In other words, it deletes a previously sent Punch In or Punch Out Setup message.

Event Start (05) and Event Stop (06)

These refer to the start/stop (ie, playback) of some continuous action (ie, an action that begins when an Event Start is received, and continues until an Event Stop is received). The Event Number refers to which action on the slave is to be started/stopped. Such actions may include playback of a specific looped waveform, a fader moving on an automated mixer, etc.

Event Start (07) and Event Stop (08) with additional info

Almost the same as the above 2 Event Types, but these have additional bytes before the final 0xF7. Such additional bytes could be for an effect unit's changing parameters, the volume level of a sound effect being adjusted, etc. The additional info should be nibblized with the lowest bits first. For example, if the Note On message 0x91 0x46 0x7F was to be encoded in some additional info bytes, they would be 01 09 06 04 0F 07.

Delete Event Start (09) and Delete Event Stop (0A)

Deletes the Event Start or Event Stop (with the matching Event Number and SMPTE Time) from the slave's event list. In other words, it deletes a previously sent Event Start or Event Stop Setup message (either the Types without additional info, or with additional info).

Cue Point (0B)

Sets an action to be triggered (ie, an action that does something once and automatically stops afterward) or a marker at the specified SMPTE time. These include a "hit" point for a sound effect, a marker for an edit point, etc. The Event Number should represent the action or marker. For example, Event Number 3 could be to trigger a car crash sound effect. Then, several car crashes could be specified by sending several Cue Point Setup messages, each with Event Number 3, but different SMPTE times.

Cue Point (0C) with additional info

Like the above, but this message may have additional bytes before the final 0xF7. Such additional bytes could be for an effect unit's parameters, the volume level of a sound effect, etc. The additional info should be nibblized with the lowest bits first.

Delete Cue Point (0D)

Deletes one of the preceding 2 Setup messages (with the same Event Number and SMPTE time) from the slave's event list.

Event Name (0E) with additional info

This assigns an ascii name to the event with the matching Event Number and SMPTE time. It for the musician's point of reference. The additional info bytes are the ascii name. For a newline character, include both a carriage return (0x0A)

and line feed (0x0D). The ascii bytes are nibblized. For example, ascii 'A' (0x41) becomes the two bytes, 0x01 0x04.

To summarize the interaction between master and slave depending upon “play mode”:

Play Mode

The master is in normal play at normal or vari-speed rates. The master is sending Quarter Frame messages to the slave. The messages are in ascending order, starting with 0xF1 0x0n and ending with 0xF1 0x7n. If the master is capable of reverse play, then the messages are sent in reverse, starting with 0xF1 0x7n and ending with 0xF1 0x0n.

Cue Mode

The master is being “rocked” or “cued” by hand. For example, a tape machine may have the tape still in contact with the playback head so that the musician can cue the contents of the tape to a specific point. The master is sending Quarter Frame messages to the slave. The messages are in ascending order, starting with 0xF1 0x0n and ending with 0xF1 0x7n. If the master is playing in a reverse direction, then the messages are sent in reverse, starting with 0xF1 0x7n and ending with 0xF1 0x0n. Because the musician may be changing the tape direction rapidly, the order of the Quarter Frames must change along with the tape direction.

Fast Forward or Rewind Mode

The master is rewinding or fast forwarding tape. No contact is made with the playback head. So, no cueing is happening. Therefore, the master only need send the slave periodic Full Frame messages at regular intervals as a rough indication of the master’s position. The SMPTE time indicated by the last Full Frame message actually takes affect upon the reception of the next Quarter Frame message (ie, when Play Mode resumes).