

Standard MIDI Files 1.0

Introduction

The document outlines the specification for MIDI Files. The purpose of MIDI Files is to provide a way of interchanging time-stamped MIDI data between different programs on the same or different computers. One of the primary design goals is compact representation, which makes it very appropriate for a disk-based file format, but which might make it inappropriate for storing in memory for quick access by a sequencer program. (It can be easily converted to a quickly-accessible format on the fly as files are read in or written out.) It is not intended to replace the normal file format of any program, though it could be used for this purpose if desired.

MIDI Files contain one or more MIDI streams, with time information for each event. Song, sequence, and track structures, tempo and time signature information, are all supported. Track names and other descriptive information may be stored with the MIDI data. This format supports multiple tracks and multiple sequences so that if the user of a program which supports multiple tracks intends to move a file to another one, this format can allow that to happen.

This spec defines the 8-bit binary data stream used in the file. The data can be stored in a binary file, nibbleized, 7-bit-ized for efficient MIDI transmission, converted to Hex ASCII, or translated symbolically to a printable text file. This spec addresses what's in the 8-bit stream. It does not address how a MIDI File will be transmitted over MIDI. It is the general feeling that a MIDI transmission protocol will be developed for files in general and MIDI Files will use this scheme.

Sequences, Tracks, Chunks: File Block Structure

Conventions

In this document, bit 0 means the least significant bit of a byte, and bit 7 is the most significant.

Some numbers in MIDI Files are represented in a form called a variable-length quantity. These numbers are represented 7 bits per byte, most significant bits first. All bytes except the last have bit 7 set, and the last byte has bit 7 clear. If the number is between 0 and 127, it is thus represented exactly as one byte.

Here are some examples of numbers represented as variable-length quantities:

Number (hex)	Representation (hex)
00000000	00
00000040	40
0000007F	7F
00000080	81 00
00002000	C0 00
00003FFF	FF 7F
00004000	81 80 00
00100000	C0 80 00
001FFFFF	FF FF 7F
00200000	81 80 80 00
08000000	C0 80 80 00
0FFFFFFF	FF FF FF 7F

The largest number which is allowed is 0xFFFFFFFF so that the variable-length representation must fit in 32 bits in a routine to write variable-length numbers. Theoretically, larger numbers are possible, but 2×10^8 96ths of a beat at a fast tempo of 500 beats per minute is four days, long enough for any delta-time!

Files

To any file system, a MIDI File is simply a series of 8-bit bytes. On the Macintosh, this byte stream is stored in the data fork of a file (with file type 'Midi'), or on the Clipboard (with data type 'Midi'). Most other computers store 8-bit byte streams in files — naming or storage conventions for those computers will be defined as required.

Chunks

MIDI Files are made up of chunks. Each chunk has a 4-character type and a 32-bit length, which is the number of bytes in the chunk. This structure allows future chunk types to be designed which may easily be ignored if encountered by a program written before the chunk type is introduced. Your programs should *expect* alien chunks and treat them as if they weren't there.

Each chunk begins with a 4-character ASCII type. It is followed by a 32-bit length, most significant byte first (a length of 6 is stored as 00 00 00 06). This length refers to the number of bytes of data which follow: the eight bytes of type and length are not included. Therefore, a chunk with a length of 6 would actually occupy 14 bytes in the disk file.

This chunk architecture is similar to that used by Electronic Arts' IFF format, and the chunks described herein could easily be placed in an IFF file. The MIDI File itself is not an IFF file: it contains no nested chunks, and chunks are not constrained to be an even number of bytes long. Converting it to an IFF file is as easy as padding odd-length chunks, and sticking the whole thing inside a FORM chunk.

MIDI Files contain two types of chunks: header chunks and track chunks. A header chunk provides a minimal amount of information pertaining to the entire MIDI file. A track chunk contains a sequential stream of MIDI data which may contain information for up to 16 MIDI channels. The concepts of multiple tracks, multiple MIDI outputs, patterns, sequences, and songs may all be implemented using several track chunks.

A MIDI file always starts with a header chunk, and is followed by one or more track chunks.

```
MThd  <length of header data>
<header data>
MTrk  <length of track data>
<track data>
MTrk  <length of track data>
<track data>
...
...
```

Chunk Descriptions

Header Chunks

The header chunk at the beginning of the file specifies some basic information about the data in the file. Here's the syntax of the complete chunk:

```
<Header Chunk> = <chunk type> <length> <format> <ntrks> <division>
```

As described above, <chunk type> is the four ASCII characters 'MThd'; <length> is a 32-bit representation of the number 6 (high byte first).

The data section contains three 16-bit words, stored most-significant byte first.

The first word, <format>, specifies the overall organization of the file. Only three values of <format> are specified:

- | | |
|---|---|
| 0 | the file contains a single multi-channel track |
| 1 | the file contains one or more simultaneous tracks (or MIDI outputs) of a sequence |
| 2 | the file contains one or more sequentially independent single-track patterns |

More information about these formats is provided below.

The next word, <ntrks>, is the number of track chunks in the file. It will always be 1 for a format 0 file.

The third word, <division>, specifies the meaning of the delta-times. It has two formats, one for metrical time, and one for time-code-based time:

0	ticks per quarter-note	
1	negative SMPTE format	ticks per frame
15	14	8 7 0

If bit 15 of <division> is a zero, the bits 14 thru 0 represent the number of delta-time "ticks" which make up a quarter-note. For instance, if <division> is 96, then a time interval of an eighth-note between two events in the file would be 48.

If bit 15 of <division> is a one, delta-times in a file correspond to subdivisions of a second, in a way consistent with SMPTE and MIDI time code. Bits 14 thru 8 contain one of the four values -24, -25, -29, or -30, corresponding to the four standard SMPTE and MIDI time code formats (-29 corresponds to 30 drop frame), and represents the number of frames per second. These negative numbers are stored in two's complement form. The second byte (stored positive) is the resolution within a frame: typical values may be 4 (MIDI time code resolution), 8, 10, 80 (bit resolution), or 100. This system allows exact specification of time-code-based tracks, but also allows millisecond-based tracks by specifying 25 frames/sec and a resolution of 40 units per frame. If the events in a file are stored with bit resolution of thirty-frame time code, the division word would be E250 hex.

Formats 0, 1, and 2

A Format 0 file has a header chunk followed by one track chunk. It is the most interchangeable representation of data. It is very useful for a simple single-track player in a program which needs to make synthesizers make sounds, but which is primarily concerned with something else such as mixers or sound effect boxes. It is very desirable to be able to produce such a format, even if your program is track-based, in order to work with these simple programs. On the other hand, perhaps someone will write a format conversion from format 1 to format 0 which might be so easy to use in some setting that it would save you the trouble of putting it into your program.

A Format 1 or 2 file has a header chunk followed by one or more track chunks. Programs which support several simultaneous tracks should be able to save and read data in format 1, a vertically one-dimensional form, that is, as a collection of tracks. Programs which support several independent patterns should be able to save and read data in format 2, a horizontally one-dimensional form. Providing these minimum capabilities will ensure maximum interchangeability.

In a MIDI system with a computer and a SMPTE synchronizer which uses Song Pointer and Timing Clock, tempo maps (which describe the tempo throughout the track, and may also include time signature information, so that the bar number may be derived) are generally created on the computer. To use them with the synchronizer, it is necessary to transfer them from the computer. To make it easy for the synchronizer to extract this data from a MIDI File, tempo information should always be stored in the first MTrk chunk. For a format 0 file, the tempo will be scattered through the track and the tempo map reader should ignore the intervening events; for a format 1 file, the tempo map must be stored as the first track. It is polite to a tempo map reader to offer your user the ability to make a format 0 file with just the tempo, unless you can use format 1.

All MIDI Files should specify tempo and time signature. If they don't, the time signature is assumed to be 4/4, and the tempo 120 beats per minute. In format 0, these meta-events should occur at least at the beginning of the single multi-channel track. In format 1, these meta-events should be contained in the first track. In format 2, each of the temporally independent patterns should contain at least initial time signature and tempo information.

We may decide to define other format IDs to support other structures. A program encountering an unknown format ID may still read other MTrk chunks it finds from the file, as format 1 or 2, if its user can make sense of them and arrange them into some other structure if appropriate. Also, more parameters may be added to the MThd chunk in the future: it is important to read and honor the length, even if it is longer than 6.

Track Chunks

The track chunks (type MTrk) are where actual song data is stored. Each track chunk is simply a stream of MIDI events (and non-MIDI events), preceded by delta-time values. The format for Track Chunks (described below) is exactly the same for all three formats (0, 1, and 2: see "Header Chunk" above) of MIDI Files.

Here is the syntax of an MTrk chunk (the + means "one or more": at least one MTrk event must be present):

```
<Track Chunk> = <chunk type> <length> <MTrk event>+
```

The syntax of an MTrk event is very simple:

```
<MTrk event> = <delta-time> <event>
```

`<delta-time>` is stored as a variable-length quantity. It represents the amount of time before the following event. If the first event in a track occurs at the very beginning of a track, or if two events occur simultaneously, a delta-time of zero is used. Delta-times are *always* present. (*Not* storing delta-times of 0 requires at least two bytes for any other value, and most delta-times *aren't* zero.) Delta-time is in ticks as specified in the header chunk.

```
<event> = <MIDI event> | <sysex event> | <meta-event>
```

`<MIDI event>` is any MIDI channel message. Running status is used: status bytes of MIDI channel messages may be omitted if the preceding event is a MIDI channel message with the same status. The first event in each MTrk chunk must specify status. Delta-time is *not* considered an event itself: it is an integral part of the syntax for an MTrk event. Notice that running status occurs *across* delta-times.

`<sysex event>` is used to specify a MIDI system exclusive message, either as one unit or in packets, or as an "escape" to specify any arbitrary bytes to be transmitted. A normal complete system exclusive message is stored in a MIDI File in this way:

```
F0 <length> <bytes to be transmitted after F0>
```

The length is stored as a variable-length quantity. It specifies the number of bytes which follow it, not including the F0 or the length itself. For instance, the transmitted message F0 43 12 00 07 F7 would be stored in a MIDI file as F0 05 43 12 00 07 F7. It is required to include the F7 at the end so that the reader of the MIDI file knows that it has read the entire message.

Another form of sysex event is provided which does not imply that an F0 should be transmitted. This may be used as an "escape" to provide for the transmission of things which would not otherwise be legal, including system realtime messages, song pointer or select, MIDI Time Code, etc. This uses the F7 code:

```
F7 <length> <all bytes to be transmitted>
```

Unfortunately, some synthesizer manufacturers specify that their system exclusive messages are to be transmitted as little packets. Each packet is only part of an entire syntactical system exclusive message, but the times they are transmitted at are important. Examples of this are the bytes sent in a CZ patch dump, or the FB-01's "system exclusive mode" in which microtonal data can be transmitted. The F0 and F7 sysex events may be used together to break up syntactically complete system exclusive messages into timed packets.

An F0 sysex event is used for the first packet in a series — it is a message in which the F0 should be transmitted. An F7 sysex event is used for the remainder of the packets, which do not begin with F0. (Of course, the F7 is not considered part of the system exclusive message).

A syntactic system exclusive message must always end with an F7, even if the real-life device didn't send one, so that you know when you've reached the end of an entire sysex message without looking ahead to the next event in the MIDI file. If it's stored in one complete F0 sysex event, the last byte must be an F7. If it is broken up into packets, the last byte of the last packet must be an F7. There also must not be any transmittable MIDI events in between the packets of a multi-packet system exclusive message. This principle is illustrated in the paragraph below.

Here is an example of a multi-packet system exclusive message: suppose the bytes F0 43 12 00 were to be sent, followed by a 200-tick delay, followed by the bytes 43 12 00 43 12 00, followed by a 100-tick delay, followed by the bytes 43 12 00 F7, this would be in the MIDI File:

```
F0 03 43 12 00
81 48
F7 06 43 12 00 43 12 00
64
F7 04 43 12 00 F7
```

200-tick delta-time
100-tick delta-time

When reading a MIDI File, and an F7 sysex event is encountered without a preceding F0 sysex event to start a multi-packet system exclusive message sequence, it should be presumed that the F7 event is being used as an "escape". In this case, it is not necessary that it end with an F7, unless it is desired that the F7 be transmitted.

<meta-event> specifies non-MIDI information useful to this format or to sequencers, with this syntax:

```
FF <type> <length> <bytes>
```

All meta-events begin with FF, then have an event type byte (which is always less than 128), and then have the length of the data stored as a variable-length quantity, and then the data itself. If there is no data, the length is 0. As with chunks, future meta-events may be designed which may not be known to existing programs, so programs must properly ignore meta-events which they do not recognize, and indeed, should *expect* to see them. Programs must never ignore the length of a meta-event which they do recognize, and they shouldn't be surprised if it's bigger than they expected. If so, they must ignore everything past what they know about. However, they must not add anything of their own to the end of a meta-event.

Sysex events and meta-events cancel any running status which was in effect. Running status does not apply to and may not be used for these messages.

Meta-Events

A few meta-events are defined herein. It is not required for every program to support every meta-event.

In the syntax descriptions for each of the meta-events a set of conventions is used to describe parameters of the events. The FF which begins each event, the type of each event, and the lengths of events which do not have a variable amount of data are given directly in hexadecimal. A notation such as dd or se, which consists of two lower-case letters, mnemonically represents an 8-bit value. Four identical lower-case letters such as wwww refer to a 16-bit value, stored most-significant-byte first. Six identical lower-case letters such as tttttt refer to a 24-bit value, stored most-significant-byte first. The notation len refers to the length portion of the meta-event syntax, that is, a number, stored as a variable-length quantity, which specifies how many data bytes follow it in the meta-event. The notations text and data refer to however many bytes of (possibly text) data were just specified by the length.

In general, meta-events in a track which occur at the same time may occur in any order. If a copyright event is used, it should be placed as early as possible in the file, so it will be noticed easily. Sequence Number and Sequence/Track Name events, if present, must appear at time 0. An end-of-track event must occur as the last event in the track.

Meta-events initially defined include:

FF 00 02 ssss

Sequence Number

This optional event, which must occur at the beginning of a track, before any nonzero delta-times, and before any transmittable MIDI events, specifies the number of a sequence. In a format 2 MIDI file, it is used to identify each "pattern" so that a "song" sequence using the Cue message to refer to the patterns. If the ID numbers are omitted, the sequences' locations in order in the file are used as defaults. In a format 0 or 1 MIDI file, which only contain one sequence, this number should be contained in the first (or only) track. If transfer of several multitrack sequences is required, this must be done as a group of format 1 files, each with a different sequence number.

FF 01 len text

Text Event

Any amount of text describing anything. It is a good idea to put a text event right at the beginning of a track, with the name of the track, a description of its intended orchestration, and any other information which the user wants to put there. Text events may also occur at other times in a track, to be used as lyrics, or descriptions of cue points. The text in this event should be printable ASCII characters for maximum interchange. However, other character codes using the high-order bit may be used for interchange of files between different programs on the same computer which supports an extended character set. Programs on a computer which does not support non-ASCII characters should ignore those characters.

Meta event types 01 through 0F are reserved for various types of text events, each of which meets the specification of text events(above) but is used for a different purpose:

FF 02 len text

Copyright Notice

Contains a copyright notice as printable ASCII text. The notice should contain the characters (C), the year of the copyright, and the owner of the copyright. If several pieces of music are in the same MIDI file, all of the copyright notices should be placed together in this event so that it will be at the beginning of the file. This event should be the first event in the first track chunk, at time 0.

FF 03 len text	Sequence/Track Name
	If in a format 0 track, or the first track in a format 1 file, the name of the sequence. Otherwise, the name of the track.
FF 04 len text	Instrument Name
	A description of the type of instrumentation to be used in that track. May be used with the MIDI Prefix meta-event to specify which MIDI channel the description applies to, or the channel may be specified as text in the event itself.
FF 05 len text	Lyric
	A lyric to be sung. Generally, each syllable will be a separate lyric event which begins at the event's time.
FF 06 len text	Marker
	Normally in a format 0 track, or the first track in a format 1 file. The name of that point in the sequence, such as a rehearsal letter or section name ("First Verse", etc.).
FF 07 len text	Cue Point
	A description of something happening on a film or video screen or stage at that point in the musical score ("Car crashes into house", "curtain opens", "she slaps his face", etc.)
FF 20 01 cc	MIDI Channel Prefix
	The MIDI channel (0-15) contained in this event may be used to associate a MIDI channel with all events which follow, including System Exclusive and meta-events. This channel is "effective" until the next normal MIDI event (which contains a channel) or the next MIDI Channel Prefix meta-event. If MIDI channels refer to "tracks", this message may help jam several tracks into a format 0 file, keeping their non-MIDI data associated with a track. This capability is also present in Yamaha's ESEQ file format.
FF 2F 00	End of Track
	This event is <i>not</i> optional. It is included so that an exact ending point may be specified for the track, so that it has an exact length, which is necessary for tracks which are looped or concatenated.
FF 51 03 tttttt	Set Tempo , in microseconds per MIDI quarter-note
	This event indicates a tempo change. Another way of putting "microseconds per quarter-note" is "24ths of a microsecond per MIDI clock". Representing tempos as time per beat instead of beat per time allows absolutely exact long-term synchronization with a time-based sync protocol such as SMPTE time code or MIDI time code. This amount of accuracy provided by this tempo resolution allows a four-minute piece at 120 beats per minute to be accurate within 500 usec at the end of the piece. Ideally, these events should only occur where MIDI clocks would be located — this convention is intended to guarantee, or at least increase the likelihood, of compatibility with other synchronization devices so that a time signature/tempo map stored in this format may easily be transferred to another device.
FF 54 05 hr mn se fr ff	SMPTE Offset
	This event, if present, designates the SMPTE time at which the track chunk is supposed to start. It should be present at the beginning of the track, that is, before any nonzero delta-times, and before any transmittable MIDI events. The hour <i>must</i> be encoded with the SMPTE format, just as it is in MIDI Time Code. In a format 1 file, the SMPTE Offset must be stored with the tempo map, and has no meaning in any of the other

tracks. The ff field contains fractional frames, in 100ths of a frame, even in SMPTE-based tracks which specify a different frame subdivision for delta-times.

FF 58 04 nn dd cc bb **Time Signature**

The time signature is expressed as four numbers. nn and dd represent the numerator and denominator of the time signature as it would be notated. The denominator is a negative power of two: 2 represents a quarter-note, 3 represents an eighth-note, etc. The cc parameter expresses the number of **MIDI clocks** in a metronome click. The bb parameter expresses the number of notated 32nd-notes in what MIDI thinks of as a quarter-note (24 MIDI Clocks). This was added because there are already multiple programs which allow the user to specify that what MIDI thinks of as a quarter-note (24 clocks) is to be notated as, or related to in terms of, something else.

Therefore, the complete event for 6/8 time, where the metronome clicks every three eighth-notes, but there are 24 clocks per quarter-note, 72 to the bar, would be (*in hex*):

FF 58 04 06 03 24 08

That is, 6/8 time (8 is 2 to the 3rd power, so this is 06 03), 36 MIDI clocks per dotted-quarter (24 hex!), and eight notated 32nd-notes per MIDI quarter note.

FF 59 02 sf mi **Key Signature**

sf = -7: 7 flats
sf = -1: 1 flat
sf = 0: key of C
sf = 1: 1 sharp
sf = 7: 7 sharps
mi = 0: major key
mi = 1: minor key

FF 7F len data **Sequencer-Specific Meta-Event**

Special requirements for particular sequencers may use this event type: the first byte or bytes of data is a manufacturer ID (these are one byte, or, if the first byte is 00, three bytes). As with MIDI System Exclusive, manufacturers who define something using this meta-event should publish it so that others may know how to use it. After all, this is an *interchange* format. This type of event may be used by a sequencer which elects to use this as its *only* file format; sequencers with their established feature-specific formats should probably stick to the standard features when using *this* format.

Program Fragments and Example MIDI Files

Here are some of the routines to read and write variable-length numbers in MIDI Files. These routines are in C, and use `getc` and `putc`, which read and write single 8-bit characters from/to the files `infile` and `outfile`.

```
WriteVarLen (value)
register long value;
{
    register long buffer;

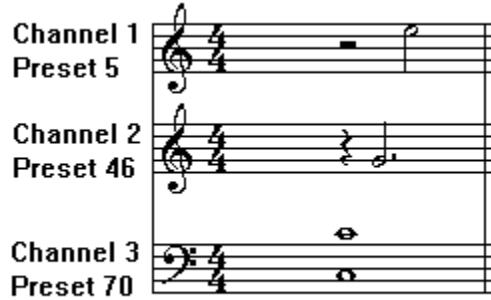
    buffer = value & 0x7f;
    while ((value >>= 7) > 0)
    {
        buffer <= 8;
        buffer |= 0x80;
        buffer += (value & 0x7f);
    }

    while (TRUE)
    {
        putc(buffer,outfile);
        if (buffer & 0x80)
            buffer >= 8;
        else
            break;
    }
}

doubleword ReadVarLen ()
{
    register doubleword value;
    register byte c;

    if ((value = getc(infile)) & 0x80)
    {
        value &= 0x7f;
        do
        {
            value = (value << 7) + ((c = getc(infile)) & 0x7f);
        } while (c & 0x80);
    }
    return (value);
}
```

As an example, MIDI Files for the following excerpt are shown below. First, a format 0 file is shown, with all information intermingled; then, a format 1 file is shown with all data separated into four tracks: one for tempo and time signature, and three for the notes. A resolution of 96 "ticks" per quarter note is used. A time signature of 4/4 and a tempo of 120, though implied, are explicitly stated.



The contents of the MIDI stream represented by this example are broken down here:

Delta Time (dec)	Event Code (hex)	Other Bytes (dec)	Comment
0	FF 58	04 04 02 24 08	4 bytes: 4/4 time, 24 MIDI clocks/click, 8 32nd notes/24 MIDI clocks
0	FF 51	03 500000	3 bytes: 500,000 µsec per quarter-note
0	C0	5	Ch. 1, Program Change 5
0	C1	46	Ch. 2, Program Change 46
0	C2	70	Ch. 3, Program Change 70
0	92	48 96	Ch. 3 Note On #48, forte
0	92	60 96	Ch. 3 Note On #60, forte
96	91	67 64	Ch. 2 Note On #67, mezzo-forte
96	90	76 32	Ch. 1 Note On #76, piano
192	82	48 64	Ch. 3 Note Off #48, standard
0	82	60 64	Ch. 3 Note Off #60, standard
0	81	67 64	Ch. 2 Note Off #67, standard
0	80	76 64	Ch. 1 Note Off #76, standard
0	FF 2F	00	Track End

The entire format 0 MIDI file contents in hex follows.

First, the header chunk:

4D 54 68 64	MThd
00 00 00 06	chunk length
00 00	format 0
00 01	one track
00 60	96 per quarter-note

Then, the track chunk. Its header, followed by the events (notice that running status is used in places):

	4D 54 72 6B 00 00 00 3B	MTrk chunk length (59)
<u>Delta-time</u>	<u>Event</u>	<u>Comments</u>
00	FF 58 04 04 02 18 08	time signature
00	FF 51 03 07 A1 20	tempo
00	C0 05	
00	C1 2E	
00	C2 46	
00	92 30 60	
00	3C 60	running status
60	91 43 40	
60	90 4C 20	
81 40	82 30 40	two-byte delta-time
00	3C 40	running status
00	81 43 40	
00	80 4C 40	
00	FF 2F 00	end of track

A format 1 representation of the file is slightly different.

First, its header chunk:

4D 54 68 64	MThd
00 00 00 06	chunk length
00 01	format 1
00 04	four tracks
00 60	96 per quarter-note

Then the track chunk for the time signature/tempo track. Its header, followed by the events:

	4D 54 72 6B 00 00 00 14	MTrk chunk length (20)
<u>Delta-time</u>	<u>Event</u>	<u>Comments</u>
00	FF 58 04 04 02 18 08	time signature
00	FF 51 03 07 A1 20	tempo
83 00	FF 2F 00	end of track

Then, the track chunk for the first music track. The MIDI convention for note on/off running status is used in this example:

	4D 54 72 6B 00 00 00 10	MTrk chunk length (16)
<u>Delta-time</u>	<u>Event</u>	<u>Comments</u>
00	C0 05	
81 40	90 4C 20	
81 40	4C 00	Running status: note on, vel = 0
00	FF 2F 00	end of track

Then, the track chunk for the second music track:

	4D 54 72 6B 00 00 00 0F	MTrk chunk length (15)
<u>Delta-time</u>	<u>Event</u>	<u>Comments</u>
00	C1 2E	
60	91 43 40	
82 20	43 00	running status
00	FF 2F 00	end of track

Then, the track chunk for the third music track:

	4D 54 72 6B 00 00 00 15	MTrk chunk length (21)
<u>Delta-time</u>	<u>Event</u>	<u>Comments</u>
00	C2 46	
00	92 30 60	
00	3C 60	running status
83 00	30 00	two-byte delta-time, running status
00	3C 00	running status
00	FF 2F 00	end of track

Calculating Delta Times:

Now here's an example of how cumulative delta time gets converted into milliseconds. In the simplest case, there are 2 pieces of information needed:

- 1) The SMF Header Chunk defines a "division" which is delta ticks per quarter note. (eg., 96 = 96 ppq) (Ref: pg. 4, SMF 1.0)
- 2) The Tempo setting, which is a non-MIDI data Meta Event typically found at time delta time 0 in the first track of an SMF. If it isn't specified, tempo is assumed to be 120 bpm. Tempo is expressed as microseconds per quarter note. (eg., 500000 = 120 bpm). (Ref: pgs. 5,9, SMF 1.0)

To convert delta-time ticks into milliseconds, you simply do a straightforward algebraic calculation:

$$\text{Time (in ms.)} = (\text{Number of Ticks}) * (\text{Tempo (uS/qn)} / \text{Div (ticks/qn)}) / 1000$$

As an example, if the Set Tempo value were 500000 uS per qn, and the Division were 96 ticks per qn, then the amount of time at 6144 Ticks into the SMF would be:

$$\begin{aligned}\text{Time} &= 6144 * (500000/96) / 1000 \\ &= 32000 \text{ milliseconds}\end{aligned}$$

The above example is a very simple case. In practice, SMFs can contain multiple Set Tempo Meta Events spaced throughout the file, and in order to calculate a correct elapsed time for any Tick, a running calculation needs to be performed.

Note that while the Time Signature is not needed to perform the above calculation, Time Signature is needed, however, if the elapsed time is desired for a particular Bar/Beat value. As with Set Tempo changes, the Time Signature can change throughout an SMF, and a running calculation is usually necessary to determine a correct elapsed time for any Bar/Beat.

General MIDI System Level 1

Published by:
The MIDI Manufacturers Association
Los Angeles, CA

PLEASE SEE MMA PUBLICATION "General MIDI System Level 1 Developer Guidelines" (1996) FOR ADDITIONAL RECOMMENDATIONS AND CLARIFICATIONS RELATED TO THIS SPECIFICATION.

MMA0007 / RP003

Copyright © 1991, 1994 MIDI Manufacturers Association Incorporated

ALL RIGHTS RESERVED. NO PART OF THIS DOCUMENT MAY BE REPRODUCED IN ANY FORM OR BY ANY MEANS, ELECTRONIC OR MECHANICAL, INCLUDING INFORMATION STORAGE AND RETRIEVAL SYSTEMS, WITHOUT PERMISSION IN WRITING FROM THE MIDI MANUFACTURERS ASSOCIATION.

MMA
POB 3173
La Habra CA 90632-3173

GM System - Overview

This Specification outlines a minimum MIDI configuration of a “General MIDI System” which defines a certain class of MIDI controlled sound generators. The General MIDI (or GM) System provides a high degree of compatibility between MIDI synthesizers, and adds the ability to play songs (in the form of MIDI data) created for any given MIDI synthesizer module that follows this Specification.

This class of products are intended for broad applications in the music, consumer, and entertainment markets, due to increased compatibility and unprecedented ease-of-use.

Background

Without this specification, when an end user tries to play back MIDI data on a given set of MIDI synthesizers the results can vary widely depending on what MIDI synthesizers are involved and what their capabilities are. The MIDI data has to be specially prepared for those particular synthesizers and drum machines in order to sound exactly as originally intended.

For example, the sound that plays on MIDI note messages sent over channel one/program number one is determined by the individual synthesizer manufacturer. However, there usually is little similarity between program numbers and expected timbres on today’s popular synthesizers. Other examples are the variability of pitch bend range, octave registration, or the drum note mapping.

This variety is wonderful for professional users, but can be troublesome for consumers and music authors. Therefore, it has in the past been virtually impossible to produce MIDI data that will play on all of the popular MIDI synthesizers. The data had to be made manufacturer and device specific. This has limited the availability of MIDI data titles to individual instruments or at best to those of a particular manufacturer.

The main barrier to resolving this problem is that the original MIDI specification does not specify a “minimum MIDI configuration” or set of capabilities that one could rely on being in a given synthesizer. A particular MIDI device has no idea what MIDI device is connected to the other end of its MIDI cable, and until now there was no industry-standard minimum configuration that manufacturers or authors could use as a reference.

The Solution

This General MIDI System is the solution to that problem. It describes a minimum number of voices, sound locations, drum note mapping, octave registration, pitch bend range, and controller usage, thereby defining a given set of capabilities to expect in a given synthesizer module. This mode will be identified by a logo on the instrument such as the “Compact Disc” logo shown on all devices supporting the CD standard.

General MIDI is a mode that synthesizers can be switched in and out of to provide a common “base case.” Higher end products will likely support additional modes of operation and should not be limited by General MIDI. The General MIDI Specification is also left open to further extensions (or “levels”) for advanced applications and continued improvements.

GM System - Level 1 Performance Requirements

General MIDI Sound Generator Requirements

Synthesis/Playback Technology (Sound Source Type):

- Up to the manufacturer.

Number of Voices:

- A minimum of:
 - 1) 24 fully dynamically allocated voices available simultaneously for both melodic and percussive sounds; or:
 - 2) 16 dynamically allocated voices for melody plus 8 for percussion.

MIDI Channels Supported:

- All 16 MIDI channels.
- Each channel can play a variable number of voices (polyphony).
- Each channel can play a different instrument (timbre).
- Key-based Percussion is always on channel 10.

Instruments:

- A minimum of 128 presets for Instruments (MIDI program numbers), conforming to the "GM Sound Set" (see Table 2)
- A minimum of 47 preset percussion sounds conforming to the "GM Percussion Map" (see Table 3)

General MIDI Sound Generator Recommended Hardware

- Master Volume control.
- MIDI In connector (Out and Thru connectors are optional).
- Audio Out (2 – left & right) plus Headphones connectors.

Level 1 Performance Requirements

General MIDI Protocol Implementation Requirements

Note on/Note off:

- Octave Registration: Middle C = MIDI Key 60 (3CH)
- All voices, including percussion, respond to velocity
- Voices dynamically allocated (notes/drums can re-attack using free voices)

Controller Changes:

<u>Controller #</u>	<u>Description</u>
1	Modulation
7	Volume
10	Pan
11	Expression
64	Sustain
121	Reset All Controllers
123	All Notes Off

<u>Registered Parameter #</u>	<u>Description</u>
0	Pitch Bend Sensitivity
1	Fine Tuning
2	Coarse Tuning

Channel Messages:

- Channel Pressure (Aftertouch)
- Pitch Bend (default range = ± 2 semitones)

Default Settings:

- Bend="0", Volume="100" (0-127), Controllers "normal"

GM System - Additional Messages

General MIDI System Messages

In addition to the above already-defined MIDI messages, there is a defined set of Universal Non-Real Time SysEx messages for turning General MIDI on and off at a sound module (should it have more than one mode of operation):

- Turn General MIDI System On: F0 7E <device ID> 09 01 F7

F0 7E	Universal Non-Real Time SysEx header
<device ID>	ID of target device (suggest using 7F: Broadcast)
09	sub-ID #1 = General MIDI message
01	sub-ID #2 = General MIDI On
F7	EOX

- Turn General MIDI System Off: F0 7E <device ID> 09 02 F7

F0 7E	Universal Non-Real Time SysEx header
<device ID>	ID of target device (suggest using 7F: Broadcast)
09	sub-ID #1 = General MIDI message
02	sub-ID #2 = General MIDI Off
F7	EOX

GM System - Level 1 Sound Set

General MIDI Sound Set Groupings:

(all channels except 10)

Prog #	Instrument Group	Prog #	Instrument Group
1-8	Piano	65-72	Reed
9-16	Chromatic Percussion	73-80	Pipe
17-24	Organ	81-88	Synth Lead
25-32	Guitar	89-96	Synth Pad
33-40	Bass	97-104	Synth Effects
41-48	Strings	105-112	Ethnic
49-56	Ensemble	113-120	Percussive
57-64	Brass	121-128	Sound Effects

Table 1

General MIDI Sound Set:

(MIDI Program Numbers 1 – 128; all channels except 10)

Prog # Instrument	Prog # Instrument	Prog # Instrument	Prog # Instrument
1. Acoustic Grand Piano	33. Acoustic Bass	65. Soprano Sax	97. FX 1 (rain)
2. Bright Acoustic Piano	34. Electric Bass (finger)	66. Alto Sax	98. FX 2 (soundtrack)
3. Electric Grand Piano	35. Electric Bass (pick)	67. Tenor Sax	99. FX 3 (crystal)
4. Honky-tonk Piano	36. Fretless Bass	68. Baritone Sax	100. FX 4 (atmosphere)
5. Electric Piano 1	37. Slap Bass 1	69. Oboe	101. FX 5 (brightness)
6. Electric Piano 2	38. Slap Bass 2	70. English Horn	102. FX 6 (goblins)
7. Harpsichord	39. Synth Bass 1	71. Bassoon	103. FX 7 (echoes)
8. Clavi	40. Synth Bass 2	72. Clarinet	104. FX 8 (sci-fi)
9. Celesta	41. Violin	73. Piccolo	105. Sitar
10. Glockenspiel	42. Viola	74. Flute	106. Banjo
11. Music Box	43. Cello	75. Recorder	107. Shamisen
12. Vibraphone	44. Contrabass	76. Pan Flute	108. Koto
13. Marimba	45. Tremolo Strings	77. Blown Bottle	109. Kalimba
14. Xylophone	46. Pizzicato Strings	78. Shakuhachi	110. Bag pipe
15. Tubular Bells	47. Orchestral Harp	79. Whistle	111. Fiddle
16. Dulcimer	48. Timpani	80. Ocarina	112. Shanai
17. Drawbar Organ	49. String Ensemble 1	81. Lead 1 (square)	113. Tinkle Bell
18. Percussive Organ	50. String Ensemble 2	82. Lead 2 (sawtooth)	114. Agogo
19. Rock Organ	51. SynthStrings 1	83. Lead 3 (calliope)	115. Steel Drums
20. Church Organ	52. SynthStrings 2	84. Lead 4 (chiff)	116. Woodblock
21. Reed Organ	53. Choir Aahs	85. Lead 5 (charang)	117. Taiko Drum
22. Accordion	54. Voice Oohs	86. Lead 6 (voice)	118. Melodic Tom
23. Harmonica	55. Synth Voice	87. Lead 7 (fifths)	119. Synth Drum
24. Tango Accordion	56. Orchestra Hit	88. Lead 8 (bass + lead)	120. Reverse Cymbal
25. Acoustic Guitar (nylon	57. Trumpet	89. Pad 1 (new age)	121. Guitar Fret Noise
26. Acoustic Guitar (steel)	58. Trombone	90. Pad 2 (warm)	122. Breath Noise
27. Electric Guitar (jazz)	59. Tuba	91. Pad 3 (polysynth)	123. Seashore
28. Electric Guitar (clean)	60. Muted Trumpet	92. Pad 4 (choir)	124. Bird Tweet
29. Electric Guitar (muted	61. French Horn	93. Pad 5 (bowed)	125. Telephone Ring
30. Overdriven Guitar	62. Brass Section	94. Pad 6 (metallic)	126. Helicopter
31. Distortion Guitar	63. SynthBrass 1	95. Pad 7 (halo)	127. Applause
32. Guitar harmonics	64. SynthBrass 2	96. Pad 8 (sweep)	128. Gunshot

Table 2

Level 1 Sound Set

General MIDI Percussion Map:
(Channel 10)

MIDI Key	Drum Sound	MIDI Key	Drum Sound	MIDI Key	Drum Sound
35	Acoustic Bass Drum	51	Ride Cymbal 1	67	High Agogo
36	Bass Drum 1	52	Chinese Cymbal	68	Low Agogo
37	Side Stick	53	Ride Bell	69	Cabasa
38	Acoustic Snare	54	Tambourine	70	Maracas
39	Hand Clap	55	Splash Cymbal	71	Short Whistle
40	Electric Snare	56	Cowbell	72	Long Whistle
41	Low Floor Tom	57	Crash Cymbal 2	73	Short Guiro
42	Closed Hi Hat	58	Vibraslap	74	Long Guiro
43	High Floor Tom	59	Ride Cymbal 2	75	Claves
44	Pedal Hi-Hat	60	Hi Bongo	76	Hi Wood Block
45	Low Tom	61	Low Bongo	77	Low Wood Block
46	Open Hi-Hat	62	Mute Hi Conga	78	Mute Cuica
47	Low-Mid Tom	63	Open Hi Conga	79	Open Cuica
48	Hi Mid Tom	64	Low Conga	80	Mute Triangle
49	Crash Cymbal 1	65	High Timbale	81	Open Triangle
50	High Tom	66	Low Timbale		

Table 3

GM System - Level 1 Detailed Explanation

GM Sound Set

For music authors, one of the most frustrating parts of the original MIDI specification was the lack of sound definitions. For example, where is the piano sound on this instrument (i.e. what is the program number)? The solution lies in a "sound-set-to-Program-Change-number" mapping that is specific to the General MIDI System.

This mapping only needs to take effect while operating inside a General MIDI System, and would otherwise let manufacturers organize sounds in any way they wish. In short, while operating inside a General MIDI System, this map takes effect – in any other mode, the manufacturer could present the sounds in any manner desired.

The General MIDI Sound Set (instrument and percussion maps) is shown in Tables 2 and 3. This mapping describes the MIDI Program Change numbers used to select sounds under the General MIDI System. The instrument would map these General MIDI program numbers to its own internal organization. MIDI Program numbers can be changed in real time during play.

GM Sound Definitions

General MIDI does not recommend any particular method of synthesis or playback. Each manufacturer should be free to express their own ideas and personal aesthetics when it comes to picking the exact timbres for each preset. In particular, the names in parentheses after each of the synth leads, pads, and sound effects are intended as guides.

Therefore, to promote consistency in song playback across a range of sound modules, a set of guidelines for General MIDI Score authors and Instrument manufacturers will be produced.

GM Performance Notes

For all instruments, the Modulation Wheel (Controller #1) will change the nature of the sound in the most natural (expected) way. i.e. depth of LFO; change of timbre; add more tine sound; etc.)

There are other MIDI messages currently pending in the MMA and JMSC that will become part of a General MIDI Level 2 Specification.

GM System - Logos

Rules for Application

The MMA and JMSC have approved the following design for a logo which will indicate a product that conforms to this specification.

For sound generators, GM is intended to allow the user to play back any score developed for GM without user intervention. This means a GM sound source must support all of the features described in that section without requiring any modification by the user. Only products which meet these requirements should have the GM logo.

Software, such as sequencer and notation programs, games, or other applications which create or play MIDI music, may also display a GM logo, as long as the product does not interfere with the performance of required GM data when used with a compatible sound source. For example, software which allows the user to select different sounds on playback should include a resident list of the GM sounds. In addition, any software which is GM compatible must properly play back — without modification — all controller settings and other required messages which may be found in a MIDI file or otherwise performed via MIDI.

GM Logo Variations

The logo is available from the MMA upon application and signing of a license agreement. The agreement specifies the terms, conditions and restrictions for application of a GM logo to products, packaging, and marketing materials. For details please refer to the current license agreement.



GM System Logo - This version of the logo can be applied to sound generators, applications software (games, sequencers, etc.), and scores (MIDI data) which conform to the GM System Level 1 Specification.



GM Sound Set - This version of the logo is intended for display with sound-sets (samples or patches) designed to modify a specific sound source to be GM System Level 1 compatible.

MIDI Show Control 1.1

Including 2-Phase Commit Enhancements

RP002/RP014

Copyright © 1991, 1994, 1995 MIDI Manufacturers Association

ALL RIGHTS RESERVED. NO PART OF THIS DOCUMENT MAY BE REPRODUCED IN ANY FORM OR BY ANY MEANS, ELECTRONIC OR MECHANICAL, INCLUDING INFORMATION STORAGE AND RETRIEVAL SYSTEMS, WITHOUT PERMISSION IN WRITING FROM THE MIDI MANUFACTURERS ASSOCIATION.

MMA
POB 3173
La Habra CA 90632-3173

MIDI Show Control (MSC) 1.1

1. Introduction

The purpose of MIDI Show Control is to allow MIDI systems to communicate with and to control dedicated intelligent control equipment in theatrical, live performance, multi-media, audio-visual and similar environments.

Applications may range from a simple interface through which a single lighting controller can be instructed to GO, STOP or RESUME, to complex communications with large, timed and synchronized systems utilizing many controllers of all types of performance technology.

The set of commands is modeled on the command structure of currently existing computer memory lighting, sound and show control systems. The intent is that translation between the MIDI Show Control specification and dedicated controller commands will be relatively straightforward, being based on the same operating principles. On the other hand, it has been assumed that translation will involve more than table look-up, and considerable variation will be found in data specifications and other communications details. In essence, MIDI Show Control is intended to communicate easily with devices which are designed to execute the same set or similar sets of operations.

2. General Structure

2.1. Universal System Exclusive Format

MIDI Show Control uses a single Universal Real Time System Exclusive ID number (sub-ID #1 = 02H) for all Show commands (transmissions from Controller to Controlled Device).

A guiding philosophy behind live performance control is that failures of individual Controlled Devices should not impair communications with other Controlled Devices. This principle may be implemented in either "open-loop" or "closed-loop" variations.

In open-loop control, no command responses from Controlled Device to Controller are specified or required. Open-loop control represents the most economical usage of communications bandwidth, and is fundamental to MIDI usage. MIDI Show Control includes open-loop practice for consistency with other Channel and System messages.

Closed-loop control, on the other hand, expects specified responses from Controlled Devices. Closed-loop practice requires more intelligent devices and uses more communications bandwidth, but provides more exact coordination between Controller and Controlled Devices. For closed-loop applications, MIDI Show Control uses the two-phase commit protocol, described in Sections 4.5 and 6.

In this version of Show Control, no command responses (from Controlled Device to Controller) are specified or required in order to optimize bandwidth requirements, system response time and system reliability in the event of communication difficulties with one or more Controlled Device. The guiding philosophy behind live performance control is that, as much as possible, failures of individual Controlled Devices should not impair communications with other Controlled Devices. This concept has been a part of MIDI design from the beginning and MIDI Show Control continues to use an "open-loop" design in order that standard MIDI practices may continue to be successfully utilized in applications using all types of standard Channel and System messages. However, a "closed-loop" version of Show Control has been discussed and may be created in the future.

In this document, all transmitted characters are represented in hex unless otherwise noted. The initials "msc" will be used to denote the new MIDI Show Control sub-ID #1 (= 02H). The format of a Show Control message is as follows:

```
F0 7F <device_ID> <msc> <command_format> <command> <data> F7
```

Notes:

1. No more than one command can be transmitted in a SysEx.
2. The total number of bytes in a Show Control message should not exceed 128.
3. SysEx messages must always be closed with an F7H as soon as all currently prepared information has been transmitted.

2.2. Device Identification

<device_ID> is always a DESTINATION device address.

Commands are most often addressed to one device at a time. For example, to command two lighting consoles to GO, transmit:

```
F0 7F <device_ID=1> <msc> <command_format=lighting> <GO> F7  
F0 7F <device_ID=2> <msc> <command_format=lighting> <GO> F7
```

<device_ID> values:	
00-6F	Individual IDs
70-7E	Group IDs 1-15 (optional)
7F	"All-call" ID for system wide broadcasts

Every device must be able to respond to both an individual and the "all-call" (7FH) ID. The group addressing mode is optional. A device may respond to one or more individual ID and one or more group ID. Both <device_ID> and <command_format> of a message must match the device_ID and command_format of a controlled device before the message is recognized.

If two separate controlled devices responding to the same command_format are set to respond to the same device_ID then only one message need be sent for both to respond. The "all-call" device_ID (7FH) is used for system wide "broadcasts" of identical commands to devices of the same command_format (or to all devices when used with <command_format=all-types>; see 4.1, below.)

Before fully interpreting the <device_ID> byte, parsing routines will need to look at <msc> and <command_format>, both of which follow <device_ID>, in order to first determine that the SysEx contains Show Control commands in the appropriate format.

A typical system will consist of at least one Controller attached to one or more Controlled Devices. It is possible for the same machine to be both a Controlled Device and a Controller at the same time. In this case, the machine may act as a translator, interpreter or converter of Show Control commands. According to its programmed instructions, the receipt of one type of command may result in the transmission of similar or different commands.

It is also a possibility that multiple Controller outputs could be merged and distributed to one or more Controlled Devices.

Optionally, Controlled Devices may be able to transmit (from a MIDI Out connector) MIDI Show Control commands of the type required by themselves to produce a desired result. In this condition, the Controlled Device will be transmitting a valid MIDI Show Control command but may not necessarily be doing so as a Controller. This is useful when the Controller has the ability (through MIDI In) to capture valid MIDI Show Control messages in order to conveniently create and edit the database of messages needed for the performances being controlled. In this case, the Controlled Device will be transmitting to the Controller, but only for the purposes of capturing messages to store and retransmit during performance.

Another application allowed by the transmission of Show Control commands by Controlled Devices is the slaving of multiple Devices of similar type. For example, if a dedicated lighting console transmits a Show Control command to "GO" when its GO button is pressed, then any other dedicated lighting console that obeys MIDI Show Control commands will also GO if it receives MIDI from the first console. In this way, many Controlled Devices may be controlled by another Controlled Device acting as the Controller. Interconnection would follow the same pattern as the normal Controller to Controlled Device arrangement.

2.3. Command_Formats

A command_format is a message byte from a Controller to a Controlled Device which identifies the format of the following Command byte. Each command_format has a format code between 01H and 7FH, and must be followed by a valid command byte. (Command_format 00H is reserved for extensions, and not all codes are currently defined.)

2.4. Commands

A command is a message byte from a Controller to a Controlled Device. Each command has a command code between 01H and 7FH, and may be followed by one or more data bytes, up to a total message length of 128 bytes. (Command 00H is reserved for extensions, and not all codes are currently defined.)

2.5. Extension Sets

Command_Format 00H and Command 00H are reserved for two extension sets:

00 01	1st command_format or command at 1st extension level
00 00 01	1st command_format or command at 2nd extension level

At this time, no extended functions have been defined. Nevertheless, to accommodate future extensions to MIDI Show Control, parsing routines must always check for extensions wherever command_format or command fields are encountered in data.

2.6. Data Length

The only restriction to the number of data bytes sent is that the total number of message bytes must not be more than 128. The actual data format of the transmitted message will be defined by the manufacturer of the Controlled Device. This means that the Controller (or the programmer of the Controller) must know the exact data format of the Controlled Device. This information will be manufacturer and equipment specific, so it is important that every manufacturer publish a thorough and unambiguous SysEx Implementation document.

Because this specification is intended to accommodate the needs of an extremely wide variety of equipment and industry needs, from very low cost light boards to the most complex audio/video multimedia extravaganzas, the data formats used in simpler systems will be considerably shorter and less complex than in comprehensive equipment. Data are transmitted in the order of most generic information first, with null character delimiters between each group of data bytes in order to signify the sending of progressively less generic data. For instance, simple Controlled Devices may look only at the basic data and discard the rest.

As an example, a complex Controlled Device may be able to process cue numbers with a large number of decimal point delineated subsections i.e. "235.32.7.8.654" If a Controller transmits this cue number to a simple Controlled Device that can only process numbers in the form "xxx.x", then the simple Device can either ignore these data or else respond to them in a predictable manner, such as processing cue number "235.3."

As a further example, cue number data may be transmitted calling up cue 235.3 then followed by a delimiter and data specifying cue list 36.6 and followed by a further delimiter specifying cue path 59. If the Device supports multiple cue lists but not multiple cue paths, it would process cue 235.3 in cue list 36.6 (or 36) and ignore the cue path data, simply using the current or default cue path.

Looking at the situation in the opposite manner, if simple cue number data were transmitted to a Device capable of processing all cue data, it would respond by processing that cue number in the current or default cue list using the current or default cue path.

3. Standard Specifications

Since data often contain some form of Cue Number designation, a "Standard" specification for transmission of Cue Number and related data provides consistency and saves space in the detailed data descriptions (Section 5).

3.1. Cue Numbers

When a Cue Number is sent as data, the following additional information fields may or may not be included as part of a complete "Cue Number" description: Q_list and Q_path. Q_list prescribes in which one of all currently Open Cue Lists the Q_number is to be placed or manipulated. Q_path prescribes from which Open Cue Path within all available cue storage media the Q_number is to be retrieved. The data include these information fields in the following order:

```
<Q_number> 00 <Q_list> 00 <Q_path> F7
```

Between each separate field a delimiter byte of the value 00H is placed as shown to indicate the end of the previous field and beginning of the next. It is acceptable to send only:

```
<Q_number> F7  
or  
<Q_number> 00 <Q_list> F7.
```

Controlled Devices should be able to accept more than one set of delimiter bytes, including directly before F7H, and even if no Q_number, Q_list or Q_path data are sent. Data are always terminated by F7H.

Q_number, Q_list and Q_path are expressed as ASCII numbers 0-9 (encoded as 30H-39H) with the ASCII decimal point character (2EH) used to delineate subsections. In the example above, cue 235.6 list 36.6 path 59 would be represented by the hex data:

```
32 33 35 2E 36 00 33 36 2E 36 00 35 39 F7
```

Decimal points should be separated by at least one digit, but Controlled Devices should accommodate the error of sending two or more decimal points together. Any number of decimal point delineated subsections may be used and any number of digits may be used in each subsection except that the length of the data must not cause the total length of the MIDI Show Control message to exceed 128 bytes.

Controlled Devices which do not support Q_list and (or Q_path) data must detect the 00H byte immediately after the Q_number (or Q_list) data and then discard all data until F7H is detected. Likewise, Controlled Devices which do not support the received number of decimal point delineated subsections, the received number of digits in a subsection or the total number of received characters in any field must handle the data received in a predictable and logical manner.

Controlled Devices which support Q_list and/or Q_path will normally default to the current or base Q_list and Q_path if these fields are not sent with Q_number.

For lighting applications, Q_list optionally defines the Playback or Submaster Controls (0 to 127) with which the cue corresponds.

It is highly recommended that every manufacturer publish a clear and concise description of their equipment's response to the above conditions.

3.2. Time Code Numbers

Since data often contain some form of time reference, a "Standard" specification for transmission of time provides consistency and saves space in the data descriptions.

MIDI Show Control time code and user bit specifications are entirely consistent with the formats used by MIDI Time Code and MIDI Cueing and are identical to the Standard Time Code format in MIDI Machine Control 1.0. Some extra flags have been added, but are defined such that if used in the MIDI Time Code/Cueing environment they would always be reset to zero, and so are completely transparent.

3.2.1. Standard Time Code (Types {ff} And {st})

This is the "full" form of the Time Code specification, and always contains exactly 5 bytes of data.

Two forms of Time Code subframe data are defined:

The first (labeled {ff}), contains subframe data exactly as described in the MIDI Cueing specification i.e. fractional frames measured in 1/100 frame units.

The second form (labeled {st}) substitutes time code "status" data in place of subframes. For example, when reading data from tape, it is useful to know whether these are real time code data, or simply time data updated by tachometer pulses during a high speed wind. In this case, as in other cases of "moving" time code, subframe data are practically useless, being difficult both to obtain and to transmit in a timely fashion.

hr mn sc fr (ff|st)

hr = Hours and type: 0 tt hhhh
tt = time type (bit format):

00 = 24 frame
01 = 25 frame
10 = 30 drop frame
11 = 30 frame

hhhh = hours (0-23, encoded as 00H-17H)

mn = Minutes: 0 c mmmmmm

c = color frame bit (copied from bit in time code stream):
0 = non color frame
1 = color framed code

mmmmmm = minutes (0-59, encoded as 00H-3BH)

sc = Seconds: 0 k ssssss

k = reserved - must be set to zero

ssssss = seconds (0-59, encoded as 00H-3BH)

fr = Frames, byte 5 ident and sign: 0 g i fffff

g = sign bit:

0 = positive
1 = negative (where signed time code is permitted)

i = final byte identification bit:

0 = subframes
1 = status

fffff = frames (0-29, encoded as 00H-1DH)

If final byte bit = subframes (i = 0):

ff = fractional frames: 0 bbbbbbbb (0-99, encoded as 00H-63H)

If final byte bit = status (i = 1):

st = code status bit map: 0 e v d xxxx

e = estimated code flag bit:

0 = normal time code

1 = tach or control track updated code

v = invalid code bit (ignore if e = 1):

0 = valid

1 = invalid (error or not current)

d = video field identification bit:

0 = no field information in this frame

1 = first frame in 4 or 8 field video sequence

xxxx = reserved bits - must be set to 0000

Drop Frame Notes:

1. When writing time code data, the drop-frame or non-drop-frame status of the data being written may be overridden by the status of the Controlled Device (i.e. the time code from the device itself). For example, if the SET_CLOCK data are loaded with a non-drop-frame number and if the time code on the Controlled Device is drop-frame, then the SET_CLOCK data will simply be interpreted as a drop-frame number, with no attempt being made to perform any mathematical transformations.
2. Furthermore, if the above SET_CLOCK number had in fact been loaded with a non-existent drop-frame number (e.g. 00:22:00:00), then the next higher valid number would have been used (in this case, 00:22:00:02).
3. Calculation of offsets, or simply the mathematical difference between two time codes, can cause confusion when one or both of the numbers is drop-frame. For the purposes of this specification, **DROP-FRAME NUMBERS SHOULD FIRST BE CONVERTED TO NON-DROP-FRAME BEFORE OFFSET CALCULATIONS ARE PERFORMED**. Results of an offset calculation will then be expressed as non-drop-frame quantities.

To convert from drop-frame to non-drop-frame, subtract the number of frames that have been "dropped" since the reference point 00:00:00:00. For example, to convert the drop-frame number 00:22:00:02 to non-drop-frame, subtract 40 frames, giving 00:21:58:22. The number 40 is produced by the fact that 2 frames were "dropped" at each of the minute marks 01 through 09, 11 through 19, 21 and 22. (Some manufacturers will prefer to store all internal time codes as a simple quantity of frames from reference point 00:00:00:00. This reduces calculation complexity, but does require that conversions are performed at all input or output stages.)

4. Index List

4.1. Command_Formats

Command_formats fall into the categories of General, Specific and All-types. General command_formats have a least significant nibble equal to 0, except for lighting which is 01H. Specific command_formats are related to the General command_format with the most significant nibble of the same value, but represent a more restricted range of functions within the format.

Command_format "All-types" (7FH) is used for system wide "broadcasts" of identical commands to devices of the same device_ID (or to all devices when used with <device_ID>=All-Call; see 2.2, above). For example, use of the All-types command_format along with the All-call device_ID allows a complete system to be RESET with a single message.

Controlled Devices will normally respond to only one command_format besides All-types. Occasionally, more complex control systems will respond to more than one command_format since they will be in control of more than one technical performance element. Controllers, of course, should normally be able to create and send commands in all command_formats, otherwise their usefulness will be limited.

Although it can be seen that a wide variety of potentially dangerous and life-threatening performance processes may be under MIDI Show Control, the intent of this specification is to allow the user considerably more exacting and precise control over the type of command_format and command which will result in the desired result than normally may be provided in a non-electronic cueing situation. The major advantages to the use of MIDI Show Control in these conditions are:

1. Less likelihood of errors in cueing. Digital communications can be demonstrated to be extremely reliable in repetitive duty conditions; much more so than tired or inexperienced stagehands.
2. More precise timing. Likewise, digital communications and computer control can be consistently accurate in automatic timing sequences and exactly as accurate as their human operators when under manual control.

IN NO WAY IS THIS SPECIFICATION INTENDED TO REPLACE ANY ASPECT OF NORMAL PERFORMANCE SAFETY WHICH IS EITHER REQUIRED OR MAKES GOOD SENSE WHEN DANGEROUS EQUIPMENT IS IN USE. MANUAL CONTROLS SUCH AS EMERGENCY STOPS, DEADMAN SWITCHES, CONFIRMATION ENABLE CONTROLS OR LIKE SAFETY DEVICES SHALL BE USED FOR MAXIMUM SAFETY.

AUTOMATIC SAFETY DEVICES SUCH AS LIMIT SWITCHES, PROXIMITY SENSORS, GAS DETECTORS, INFRARED CAMERAS AND PRESSURE AND MOTION DETECTORS SHALL BE USED FOR MAXIMUM SAFETY. MIDI SHOW CONTROL IS NOT INTENDED TO TELL DANGEROUS EQUIPMENT WHEN IT IS SAFE TO GO: IT IS ONLY INTENDED TO SIGNAL WHAT IS DESIRED IF ALL CONDITIONS ARE ACCEPTABLE AND IDEAL FOR SAFE PERFORMANCE. ONLY PROPERLY DESIGNED SAFETY SYSTEMS AND TRAINED SAFETY PERSONNEL CAN ESTABLISH IF CONDITIONS ARE ACCEPTABLE AND IDEAL AT ANY TIME.

TWO-PHASE COMMIT METHODOLOGY IS EXCEPTIONALLY ERROR-FREE AND CAN BE UTILIZED TO ADD SAFETY FEATURES TO SHOW CONTROL SYSTEMS; HOWEVER THIS MUST STILL BE IMPLEMENTED ACCORDING TO THE PARAMETERS OF THIS SPECIFICATION AND ONLY IN ADDITION TO THE ABOVE SAFETY CAVEATS.

Hex	command_format	Hex	command_format
00	reserved for extensions	40	Projection (General)
01	Lighting (General Category)	41	Film Projectors
02	Moving Lights	42	Slide Projectors
03	Color Changers	43	Video Projectors
04	Strobes	44	Dissolvers
05	Lasers	45	Shutter Controls
06	Chasers		
10	Sound (General Category)	50	Process Control (Gen.)
11	Music	51	Hydraulic Oil
12	CD Players	52	H ₂ O
13	EPROM Playback	53	CO ₂
14	Audio Tape Machines	54	Compressed Air
15	Intercoms	55	Natural Gas
16	Amplifiers	56	Fog
17	Audio Effects Devices	57	Smoke
18	Equalizers	58	Cracked Haze
20	Machinery (General Cat.)	60	Pyro (General Category)
21	Rigging	61	Fireworks
22	Flys	62	Explosions
23	Lifts	63	Flame
24	Turntables	64	Smoke pots
25	Trusses		
26	Robots		
27	Animation		
28	Floats		
29	Breakaways		
2A	Barges		
30	Video (General Category)	7F	All-types
31	Video Tape Machines		
32	Video Cassette Machines		
33	Video Disc Players		
34	Video Switchers		
35	Video Effects		
36	Video Character Generators		
37	Video Still Stores		
38	Video Monitors		

4.2. Recommended Minimum Sets

MIDI Show Control does not specify an absolute minimum set of commands and data which must be implemented in each device responding to a given command_format.

However, in order to ease the burden of interfacing between Controllers and Controlled Devices from different manufacturers, four RECOMMENDED MINIMUM SETS of commands and data have been created. Once a Controlled Device is specified to conform to a particular Recommended Minimum Set, then the task of designing a Controller which will successfully operate that device is considerably simplified.

The currently defined Recommended Minimum Sets are:

1. Simple Controlled Device; no time code; basic data only
2. No time code; full data capability
3. Full time code; full data capability
4. Two phase commit methodology (see Sections 4.5 and 6)

Assignment of any particular command or data to a Recommended Minimum Set may be found in the far right hand column of the Index List.

Recommended Minimum Sets are in no way intended to restrict the scope of operations supported by any device. They are offered only in the spirit of a "lowest common denominator".

4.3. General Commands

The following commands are basic to the current implementation of Memory Lighting systems and probably apply to all dedicated theatrical show control systems in a general sense. Although it is not required that Controlled Devices incorporate all of these commands, it is highly recommended:

Hex	Command	Number of data bytes	Recommended Minimum Sets
00	reserved for extensions		
01	GO	variable	1 2 3
02	STOP	variable	1 2 3
03	RESUME	variable	1 2 3
04	TIMED_GO	variable	2 3
05	LOAD	variable	2 3
06	SET	4 or 9	2 3
07	FIRE	1	2 3
08	ALL_OFF	0	2 3
09	RESTORE	0	2 3
0A	RESET	0	2 3
0B	GO_OFF	variable	2 3

4.4. Sound Commands

The following commands, in addition to the above, are basic to the current implementation of Computer Controlled Sound Memory Programming Systems and are widely used by Show Control Systems in more comprehensive applications. It is recommended that Controllers support the transmission of these commands:

Hex	command	Number of data bytes	Recommended Minimum Sets
10	GO/JAM_CLOCK	variable	3
11	STANDBY_+	variable	2 3
12	STANDBY_-	variable	2 3
13	SEQUENCE_+	variable	2 3
14	SEQUENCE_-	variable	2 3
15	START_CLOCK	variable	3
16	STOP_CLOCK	variable	3
17	ZERO_CLOCK	variable	3
18	SET_CLOCK	variable	3
19	MTCCHASE_ON	variable	3
1A	MTCCHASE_OFF	variable	3
1B	OPEN_CUE_LIST	variable	2 3
1C	CLOSE_CUE_LIST	variable	2 3
1D	OPEN_CUE_PATH	variable	2 3
1E	CLOSE_CUE_PATH	variable	2 3

4.5. Two-Phase Commit Commands

The following commands extend MIDI show control by adding two-phase commit (2PC) methodology. This methodology is not required for any form of MIDI show control operation. However, it does add data checking and error detection to the basic MIDI show control semantics. Suggested uses of these command extensions include situations where a show is being completely monitored and controlled from a central location, or performance conditions where safety requires additional checking and redundancy in the show control mechanisms.

Section 6 describes the two-phase commit methodology as applied in this specification. Review of that section is recommended before reading the specific two-phase commit message descriptions found in Section 5.

N.B. the two-phase commit methodology requires bi-directional communications. In MIDI, this means adding a data path where the MIDI OUT lines of controlled device consoles feed into the MIDI IN of the MIDI show control system that is acting as controller for show operations.

Hex	Command	Number of data bytes	Recommended Minimum Sets
20	STANDBY	variable	---4
21	STANDING_BY	variable	---4
22	GO_2PC	variable	---4
23	COMPLETE	variable	---4
24	CANCEL	variable	---4
25	CANCELLED	6	---4
26	ABORT	6	---4

5. Detailed Command And Data Descriptions

00 Reserved For Extensions

01 GO

01	GO
<Q_number>	optional; required if Q_list is sent
00	delimiter
<Q_list>	optional; required if Q_path is sent
00	delimiter
<Q_path>	optional

Starts a transition or fade to a cue. Transition time is determined by the cue in the Controlled Device. If no Cue Number is specified, the next cue in numerical sequence GOes. If a Cue Number is specified, that cue GOes. Transitions "run" until complete. If the Controller wishes to define the transition time, TIMED_GO should be sent.

In Controlled Devices with multiple Cue Lists, if no Cue Number is Specified, the next cues in numerical order and numbered identically and which are in Open Cue Lists GO. If Q_number is sent without Q_list, all cues with a number identical to Q_number and which are in Open Cue Lists GO.

02 STOP

02	STOP
<Q_number>	optional; required if Q_list is sent
00	delimiter
<Q_list>	optional; required if Q_path is sent
00	delimiter
<Q_path>	optional

Halts currently running transition(s). If no Cue Number is specified, all running transitions STOP. If a Cue Number is specified, only that single, specific transition STOPs, leaving all others unchanged.

03 RESUME

03	RESUME
<Q_number>	optional; required if Q_list is sent
00	delimiter
<Q_list>	optional; required if Q_path is sent
00	delimiter
<Q_path>	optional

Causes STOPped transition(s) to continue running. If no Cue Number is specified, all STOPped transitions RESUME. If a Cue Number is specified, only that transition RESUMES, leaving all others unchanged.

04 TIMED_GO

04	TIMED_GO
hr mn sc fr ff	Standard Time Specification
<Q_number>	optional; required if Q_list is sent
00	delimiter
<Q_list>	optional; required if Q_path is sent
00	delimiter
<Q_path>	optional

Starts a timed transition or fade to a cue. If no Cue Number is specified, the next cue in numerical sequence GOes. If a Cue Number is specified, that cue GOes. Transitions "run" until complete.

Time is Standard Time Specification with subframes (type {ff}), providing anything from "instant" to 24 hour transitions. If a Controlled Device does not support TIMED_GO it should GO instead, ignoring the time data but processing Cue Number data normally. If the transition time desired is the preexisting default cue time, GO should be sent instead of TIMED_GO.

Rules for Controlled Devices with multiple Cue Lists are the same as for GO, above.

05 LOAD

05	LOAD
<Q_number>	required
00	delimiter
<Q_list>	optional; required if Q_path is sent
00	delimiter
<Q_path>	optional

Places a cue into a standby position. Cue Number must be specified. LOAD is useful when the cue desired takes a finite time to access. LOAD is sent in advance so that the cue will GO instantly.

In Controlled Devices with multiple Cue Lists, if Q_number is sent without Q_list, all cues with a number identical to Q_number and which are in Open Cue Lists LOAD to standby.

06 SET

06	SET
cc cc	Generic Control Number, LSB first
vv vv	Generic Control Value, LSB first
hr mn sc fr ff	Standard Time Specification, optional

Standard Generic Control Numbers for Lighting

0-127	Sub masters	224-255	Chase sequence masters
128-129	Masters of the first playback	256-287	Chase sequence speed masters
130-131	Masters of the second playback	510	Grand Master for all channels
...	(etc.)	511	General speed controller for all fades
190-191	Masters of the 32nd playback	512-1023	Individual channel levels
192-223	Speed controllers for the 32 playbacks		

Defines the value of a Generic Control. The Generic Control and its value are each specified by a 14 bit number. A Controlled Device may treat virtually any of its variables, attributes, rates, levels, modes, functions, effects, subs, channels, switches, etc. as Generic Controls which may be sent values via SET. Optionally, the time it takes the Generic Control to achieve its value may be sent.

Time is Standard Time Specification with subframes (type {ff}), providing anything from "instant" to 24 hour transitions. If a Controlled Device does not support times in SET, it should ignore time data.

07 FIRE

07	FIRE
mm	Macro Number

Triggers a pre-programmed keyboard Macro. The Macro is defined by a 7 bit number. The Macros themselves are either programmed at the Controlled Device, or loaded via MIDI file dump facilities using the ASCII Cue Data format or any method applicable to the Device.

08 ALL_OFF

08	ALL_OFF
----	---------

Independently turns all functions and outputs off without changing the control settings. Operating status prior to ALL_OFF may be reestablished by RESTORE.

09 RESTORE

09	RESTORE
----	---------

Reestablishes operating status to exactly as it was prior to ALL_OFF.

0A RESET

0A	RESET
----	-------

Terminates all running cues, setting all timed functions to an initialized state equivalent to a newly powered-up condition and loads the first cue of each applicable cue list into the appropriate standby positions. In other words, RESET stops the show without arbitrarily changing any control values and loads the top of the show to standby.

It should be decided by the manufacturer of the Controlled Device whether or not RESET should automatically open all CLOSED_CUE_LISTs and CLOSED_CUE_PATHs and this decision should be stated clearly in the device's MIDI Implementation documentation.

0B GO_OFF

0B	GO_OFF
<Q_number>	optional; required if Q_list is sent
00	delimiter
<Q_list>	optional; required if Q_path is sent
00	delimiter
<Q_path>	optional

Starts a transition or fade of a cue to the off state. Transition time is determined by the cue in the Controlled Device.

If no Cue Number is specified, the current cue GOes OFF. If a Cue Number is specified, that cue GOes OFF.

In Controlled Devices with multiple Cue Lists, if no Cue Number is Specified, all currently active cues in Open Cue Lists GO OFF. If Q_number is sent without Q_list, all cues with a number identical to Q_number and which are in Open Cue Lists GO OFF.

For compatibility with Controlled Devices which do not automatically replace an existing cue with a new cue upon receipt of the GO command, Controllers should optionally prompt the programmer to simultaneously create a GO_OFF command.

10 GO/JAM_CLOCK

10	GO/JAM_CLOCK
<Q_number>	optional; required if Q_list is sent
00	delimiter
<Q_list>	optional; required if Q_path is sent
00	delimiter
<Q_path>	optional

Starts a transition or fade to a cue simultaneous with forcing the clock time to the 'Go Time' if the cue is an 'Auto Follow' cue. Transition time is determined by the cue in the Controlled Device.

If no Cue Number is specified, the next cue in numerical sequence GOes and the clock of the appropriate Cue List JAMs to that cue's time. If the next cue in numerical sequence is a 'Manual' cue (i.e. if it has not been stored with a particular 'Go Time,' making it an 'Auto Follow' cue), the GO/JAM_CLOCK command is ignored.

If a Cue Number is specified, that cue GOes and the clock of the appropriate Cue List JAMs to the cue's time unless the cue is 'Manual' in which case no change occurs.

Rules for Controlled Devices with multiple Cue Lists are the same as for GO, above.

11 STANDBY_+

11	STANDBY_+
<Q_list>	optional

Places into standby position the next cue in numerical order after the cue currently in standby.

If Q_list is not sent, the Open Cue List containing the next cue in numerical order is used. If more than one Open Cue List have cues with an identical number then those cues will move to their respective standby positions.

If Q_list is sent in Standard Cue Number Form, only the next cue in the Cue List specified moves to the standby position.

12 STANDBY_-

12	STANDBY_-
<Q_list>	optional

Places into standby position the previous cue in numerical order prior to the cue currently in standby.

If Q_list is not sent, the Open Cue List containing the previous cue in numerical order is used. If more than one Open Cue List have cues with an identical number then those cues will move to their respective standby positions.

If Q_list is sent in Standard Form, only the previous cue in the Cue List specified moves to the standby position.

13 SEQUENCE_+

13	SEQUENCE_+
<Q_list>	optional

Places into standby position the next parent cue in numerical sequence after the cue currently in standby.

'Parent' refers to the integer value of the cue's number prior to the first decimal point (the "most significant number") For example, if cue 29.324.98.7 was in standby and the cues following were 29.325, 29.4, 29.7, 29.9.876, 36.7, 36.7.832, 36.8, 37., and 37.1, then cue 36.7 would be loaded to standby by SEQUENCE_+.

If Q_list is not sent, the Open Cue List containing the next cue in parental sequence is used. If more than one Open Cue List have cues with a completely identical number then those cues will move to their respective standby positions.

If Q_list is sent in Standard Form, only the next parent cue in the Cue List specified moves to the standby position.

14 SEQUENCE_-

14	SEQUENCE_-
<Q_list>	optional

Places into standby position the lowest numbered parent cue in the previous numerical sequence prior to the cue currently in standby.

'Parent' refers to the integer value of the cue's number prior to the first decimal point (the "most significant number") For example, if cue 37.4.72.18.5 was in standby and the cues preceding were 29.325, 29.4, 29.7, 29.9.876, 36.7, 36.7.832, 36.8, 37., and 37.1, then cue 36.7 would be loaded to standby by SEQUENCE_-.

If Q_list is not sent, the Open Cue List containing the previous parental sequence is used. If more than one Open Cue List have cues with identical lowest numbered parent cues in previous parental sequence then those cues will move to their respective standby positions.

If Q_list is sent in Standard Form, only the first parent cue in the previous sequence of the Cue List specified moves to the standby position.

15 START_CLOCK

15	START_CLOCK
<Q_list>	optional

Starts the 'Auto Follow' clock timer. If the clock is already running, no change occurs. The clock continues counting from the time value which it contained while it was stopped.

If Q_list is not sent, the clocks in all Open Cue Lists Start simultaneously.

If Q_list is sent in Standard Form, only the clock in that Cue List Starts.

16 STOP_CLOCK

16	STOP_CLOCK
<Q_list>	optional

Stops the 'Auto Follow' clock timer. If the clock is already stopped, no change occurs. While the clock is stopped, it retains the time value which it contained at the instant it received the STOP command.

If Q_list is not sent, the clocks in all Open Cue Lists Stop simultaneously.

If Q_list is sent in Standard Form, only the clock in that Cue List stops.

17 ZERO_CLOCK

17	ZERO_CLOCK
<Q_list>	optional

Sets the 'Auto Follow' clock timer to a value of 00:00:00:00.00, whether or not it is running. If the clock is already stopped and Zeroed, no change occurs. ZERO_CLOCK does not affect the clock's running status.

If Q_list is not sent, the clocks in all Open Cue Lists Zero simultaneously.

If Q_list is sent in Standard Form, only the clock in that Cue List Zeros.

18 SET_CLOCK

18	SET_CLOCK
hr mn sc fr ff	Standard Time Specification
<Q_list>	optional

Sets the 'Auto Follow' clock timer to a value equal to the Standard Time sent, whether or not it is running. SET_CLOCK does not affect the clock's running status.

If Q_list is not sent, the clocks in all Open Cue Lists Set simultaneously.

If Q_list is sent in Standard Form, only the clock in that Cue List Sets.

19 MTCCHASEON

19 MTCCHASEON
<Q_list> optional

Causes the 'Auto Follow' clock timer to continuously contain a value equal to incoming MIDI Time Code. If no MTC is being received when this command is received, the clock remains in its current running or stopped status until MTC is received, at which time the clock continuously exhibits the same time as MTC. If MTC becomes discontinuous, the clock continues to display the last valid MTC message value received.

If Q_list is not sent, the clocks in all Open Cue Lists Chase simultaneously. If Q_list is sent in Standard Form, only the clock in that Cue List Chases.

1A MTCCHASEOFF

1A MTCCHASEOFF
<Q_list> optional

Causes the 'Auto Follow' clock timer to cease Chasing incoming MIDI Time Code. When MTCCHASEOFF is received, the clock returns to running or stopped status according to its operating status at the instant MTCCHASEON was received.

MTCCHASEOFF does not change the clock time value; i.e. if the clock is stopped, it retains the last valid MTC message value received (or simply the most recent time in the clock register); if the clock is running, it continues to count from the most recent time in its register.

If Q_list is not sent, the clocks in all Open Cue Lists stop Chasing simultaneously.

If Q_list is sent in Standard Form, only the clock in that Cue List stops Chasing.

1B OPEN_CUE_LIST

1B OPEN_CUE_LIST
<Q_list> required

Makes a Cue List available to all other commands and includes any cues it may contain in the current show.

When OPEN_CUE_LIST is received, the specified Cue List becomes active and cues in it can be accessed by normal show requirements. Q_list in Standard Form must be sent.

If the specified Cue List is already Open or if it does not exist, no change occurs.

1C CLOSE_CUE_LIST

1C CLOSE_CUE_LIST
<Q_list> required

Makes a Cue List unavailable to all other commands and excludes any cues it may contain from the current show.

When CLOSE_CUE_LIST is received, the specified Cue List becomes inactive and cues in it cannot be accessed by normal show requirements, but the status of the cues in the list does not change. Q_list in Standard Form must be sent.

If the specified Cue List is already Closed or if it does not exist, no change occurs.

1D OPEN_CUE_PATH

1D	OPEN_CUE_PATH
<Q_path>	required

Makes a Cue Path available to all other MIDI Show Control commands and to all normal show cue path access requirements as well.

When OPEN_CUE_PATH is received, the specified Cue Path becomes active and cues in it can be accessed by the Controlled Device. Q_path in Standard Form must be sent.

If the specified Cue Path is already Open or if it does not exist, no change occurs.

1E CLOSE_CUE_PATH

1E	CLOSE_CUE_PATH
<Q_path>	required

Makes a Cue Path unavailable to all other MIDI Show Control commands and to all normal show cue path access requirements as well.

When CLOSE_CUE_PATH is received, the specified Cue Path becomes inactive and cues in it cannot be accessed by the Controlled Device. Q_path in Standard Form must be sent.

If the specified Cue Path is already Closed or if it does not exist, no change occurs.

20 STANDBY

20	STANDBY
cc cc	Checksum, LSB first
nn nn	Sequence number, LSB first
d1 d2	Data, 4 7-bit cue data values
d3 d4	(see Section 6.8)
<Q_number>	required
00	delimiter
<Q_list>	optional; required if Q_path is sent
00	delimiter
<Q_path>	optional

Normally sent by a controller. Places the identified cue in a standby (ready to execute) state. The controlled device that receives this message must respond with either a STANDING_BY or an ABORT message. If for any reason, the controlled device is unable to ready the identified cue for execution, it must respond with an ABORT message.

The d1, d2, d3, and d4 values in the STANDBY message can be used by a controlled device as additional information about the cue to be executed. Use of these values by controlled devices is optional. However, controllers must always include these values in STANDBY messages. Whenever correct d1 ... d4 values are unknown, controllers must send zeros.

The d1 ... d4 values sent in a STANDBY message must match those sent in the subsequent GO_2PC message. If this is not true, the controlled device may respond to the GO_2PC message with an ABORT message containing one of the "invalid dn cue data value" status codes. See Section 6.8 for additional information about and usage examples for the d1 ... d4 values.

The controlled device has two seconds in which to respond to a STANDBY message with a STANDING_BY message. If the controller does not receive a STANDING_BY message in this time, it proceeds as if an ABORT message was sent.

21 STANDING_BY

21	STANDING_BY
cc cc	Checksum, LSB first
nn nn	Sequence number, LSB first
hr mm sc fr ff	Max time required to execute the cue (Standard Time Specification format)
<Q_number>	optional; required if Q_list is sent
00	delimiter
<Q_list>	optional; required if Q_path is sent
00	delimiter
<Q_path>	optional

Normally sent by a controlled device. Indicates that the identified cue is ready to be executed. Note: the cue is identified by Q_number, Q_list, and Q_path. Although the d1, d2, d3, and d4 fields in the STANDBY message may modify how the cue is to be executed, they do not identify the cue.

"Ready to be executed," means, among other things, that the cue is known to the controlled device, that it is in memory or otherwise fully ready for immediate execution, and that the controlled device is fully capable of performing the actions dictated by the cue. The cue must be identified by returning the sequence number found in the STANDBY message that initiated the transaction. Optionally, the cue may be identified by both the sequence number and the <Q_number> <Q_list> and <Q_path> parameters. If both are used, they must agree completely with the values found in the STANDBY message. Otherwise, the response is treated as an ABORT message.

The STANDING_BY message includes the maximum time required to execute the cue. Later, the controller will use this time to verify that the controlled device does not fail during execution of the cue. The maximum time may be significantly larger than the actual time required, but cannot be smaller. If operator action is required as part of cue execution, then the maximum time must account for the time spent waiting for that operator action.

22 GO_2PC

22	GO_2PC
cc cc	Checksum, LSB first
nn nn	Sequence number, LSB first
d1 d2	Data, 4 7-bit cue data values
d3 d4	(see Section 6.8)
<Q_number>	required
00	delimiter
<Q_list>	optional; required if Q_path is sent
00	delimiter
<Q_path>	optional

Normally sent by a controller. Starts the execution of the identified cue. Note: the cue is identified by Q_number, Q_list, and Q_path. Although the d1, d2, d3, and d4 fields in the GO_2PC message may modify how the cue will be executed, they do not identify the cue.

When execution of the cue is complete, the controlled device responds with a COMPLETE message. If for any reason before or during the execution of the cue a condition requiring termination of cue execution occurs, the controlled device immediately sends an ABORT message.

Before receipt of the GO_2PC message, the controlled device must have received a STANDBY message and responded to that message with a STANDING_BY message for the cue identified by <Q_number> etc. fields in the GO_2PC message. If this is not true, the controlled device must respond to the GO_2PC message with an ABORT message containing the "not standing by" status.

The d1, d2, d3, and d4 values in the GO_2PC message can be used by a controlled device as additional information about the cue being executed. Use of these values by controlled devices is optional. However, controllers must always include these values in GO_2PC messages. Whenever correct d1 ... d4 values are unknown, controllers must send zeros.

The d1 ... d4 values sent in a GO_2PC message must match those sent in the previous STANDBY message. If this is not true, the controlled device may respond to the GO_2PC message with an ABORT message containing one of the "invalid dn cue data value" status codes. See Section 6.8 for additional information about and usage examples for the d1 ... d4 values.

The controlled device is not required to "remember" previous STANDBY - STANDING_BY exchanges forever. However, the controlled device should be capable of "remembering" enough such exchanges to make delivery of "not standing by" ABORTs due to "forgotten" exchanges extremely rare. Manufacturers should document the maximum number of STANDBY - STANDING_BY exchanges their controlled device can "remember" concurrently. A "remembered" STANDBY - STANDING_BY exchange is cleared upon receipt of the GO_2PC. Re-execution of the cue must begin with a new STANDBY - STANDING_BY exchange.

In the previously sent STANDING_BY message, the controlled device has informed the controller of the maximum time required to execute this cue. The controlled device has this much time in which to process the cue and send a COMPLETE or ABORT message. If neither of these is received in this period of time, the controller proceeds as if an ABORT message was sent.

23 COMPLETE

23	COMPLETE
cc cc	Checksum, LSB first
nn nn	Sequence number, LSB first
<Q_number>	optional; required if Q_list is sent
00	delimiter
<Q_list>	optional; required if Q_path is sent
00	delimiter
<Q_path>	optional

Normally sent by a controlled device. Indicates that the identified cue has completed execution. Note: the cue is identified by Q_number, Q_list, and Q_path. Although the d1, d2, d3, and d4 fields in the GO_2PC message may have modified how the cue was executed, they do not identify the cue.

The cue must be identified by returning the sequence number found in the GO_2PC message that initiated the transaction. Optionally, the cue may be identified by both the sequence number and the <Q_number> <Q_list> and <Q_path> parameters. If both are used, they must agree completely with the values found in the GO_2PC message. Otherwise, the response is treated as an ABORT message.

24 CANCEL

24	CANCEL
cc cc	Checksum, LSB first
nn nn	Sequence number, LSB first
<Q_number>	required
00	delimiter
<Q_list>	optional; required if Q_path is sent
00	delimiter
<Q_path>	optional

Normally sent by a controller. Indicates that the cue operation named by the <Q_number> etc. fields, should be terminated. Before receipt of the CANCEL message, at least a STANDBY and possibly a GO_2PC message also must have been received. If this is not true, the controlled device must respond with a CANCELLED message containing the "not standing by" status.

If the previous operation was a STANDBY - STANDING_BY exchange (without a GO_2PC message), the previous exchange is simply "forgotten." Re-execution of the cue must begin with a new STANDBY - STANDING_BY exchange.

If the previous operation was a GO_2PC for which a COMPLETE has not yet been sent, the operation can be:

1. Completed -- run to a normal completion,
2. Paused -- stopped but awaiting further execution,
3. Terminated -- stopped without possibility of further execution, or
4. Reversed -- returned to the state present before execution of the GO_2P message was begun.

Which of these four actions is taken depends on the equipment being controlled and the circumstances under which the CANCEL message is received. Some controlled devices may always perform one of the four actions. Other controlled devices may select the action performed under program control. Manufacturers should document the actions taken when a CANCEL message is received by their controlled devices.

If a controlled device decides to complete a canceled cue, then it must send both a CANCELLED message containing the "completing" status code and a COMPLETE message. The CANCELLED message is sent in immediate response to the CANCEL message. The COMPLETE message is sent when the cue is actually completed.

Cues that are Paused by a CANCEL message can be resumed by execution of a STANDBY - STANDING_BY - GO_2PC message exchange. Otherwise, execution of a STANDBY - STANDING_BY - GO_2PC message exchange will cause complete re-execution of the cue.

A CANCEL message must be responded to with a CANCELLED message or an ABORT message. An ABORT message response is allowed only when a "checksum error" is detected in the CANCEL message. In all other cases, the response must be a CANCELLED message. If a CANCEL message is not responded to within two seconds, processing will proceed as if an ABORT response message has been sent.

25 CANCELLED

25	CANCELLED
cc cc	Checksum, LSB first
s1 s2	Status code, [status = (s1*4)+(s2*512)]
nn nn	Sequence number, LSB first
	Standard Status codes for Canceled messages
80 04	completing
80 08	paused
80 0C	terminated
80 10	reversed
80 24	not standing by
80 28	manual override in progress

Normally sent by a controlled device. Indicates that a CANCEL message has been honored or was irrelevant. The status code indicates the disposition of the cue. Valid status codes are: "not standing by," "manual override in progress," "completing," "paused," "terminated," and "reversed."

The status code "not standing by" indicates that the cue named by <Q_number> etc. is neither "remembered" as having been represented by a prior STANDBY - STANDING_BY exchange nor currently being executed. Beyond the obvious causes of this condition, this may occur because the controlled device has already completed execution of the GO_2PC message by the time the CANCEL message is received.

Each of the "completing," "paused," "terminated," and "reversed" status codes indicate the disposition of the cue following execution of the CANCEL message. When a GO_2PC message has not been received before a CANCEL message, the status code in the CANCELLED message is always "terminated."

The "manual override in progress" status code indicates that the local operator at the controlled device has taken over control of cue execution. Devices may be designed to ignore all two-phase commit MIDI show control messages whenever the local operator is manually initiating cue actions. See Section 6.4.5 for additional information on this design feature.

The sequence number found in the CANCEL message that initiated the transaction must be used to identify the CANCEL message being responded to. No optional parameters are permitted at this time.

26 ABORT

26	ABORT
cc cc	Checksum, LSB first
s1 s2	Status code, [status = (s1*4)+(s2*512)]
nn nn	Sequence number, LSB first
	Standard Status codes for Abort messages
00 00	unknown/undefined error
80 00	checksum error
80 20	*timeout
80 24	not standing by
80 28	manual override initiated
80 30	manual override in progress
80 40	deadman interlock not established
80 44	required safety interlock not established
80 50	unknown <Q_number>
80 54	unknown <Q_list>
80 58	unknown <Q_path>
80 5C	too many cues active

80 60	cue out of sequence
80 64	invalid d1 cue data value
80 68	invalid d2 cue data value
80 6C	invalid d3 cue data value
80 70	invalid d4 cue data value
80 90	manual cueing of playback medium required
80 A0	power failure in controlled device subsystem
80 B0	reading new show cues from disk
10 04	(meaning is dependent on Command_Format)
10 08	(meaning is dependent on Command_Format)
10 0C	(meaning is dependent on Command_Format)
10 10	(meaning is dependent on Command_Format)
10 14	(meaning is dependent on Command_Format)
10 18	(meaning is dependent on Command_Format)
10 1C	(meaning is dependent on Command_Format)
11 04	(meaning is dependent on Command_Format)
11 08	(meaning is dependent on Command_Format)
12 04	(meaning is dependent on Command_Format)

Normally sent by a controlled device. Indicates a failure to execute a STANDBY, GO_2PC, or CANCEL message. The status code indicates the most severe reason for the failure to execute the previous message. See Section 6.7 for a complete discussion of status codes. The sequence number found in the message that initiated the transaction must be used to identify the STANDBY, GO_2PC, or CANCEL message being ABORTed. N.B. there may be additional, less severe, reasons why the message could not be executed. Thus, correcting the error condition reported by the status code may not be sufficient to allow execution.

Status code severity is related to the ease with which the condition can be corrected. For example, the "deadman interlock not established" status code is less severe than the "motor failure" status code. The former can be corrected by immediate human action. Correcting the latter condition may require disassembly of the motor.

The "manual override in progress" status code indicates that the local operator at the controlled device has taken over control of cue execution. Devices may be designed to ignore all two-phase commit MIDI show control messages whenever the local operator is manually initiating cue actions. See Section 6.4.5 for additional information on this design feature.

6. Two-Phase Commit Details

Two-phase commit (2PC) is a transaction coordination methodology. It is a solution to the generals' paradox, which essentially asks the question, "How can several generals conspire to communicate in a way that guarantees that they all march together or none march at all, regardless of errors in their communications channels?" The two-phase commit solution to the generals' paradox involves two distinct phases of communications. In the first phase, all parties agree on what is to be done. In the second phase, all parties initiate the previously agreed upon actions and report their results.

Two-phase commit methodology is very similar to the cue coordination methodology employed by theatrical technicians on vocal cue calling systems. The first phase is the 'standby' phase. The second phase is the 'go' phase.

6.1. Controllers And Controlled Devices

In two-phase commit communications, Controllers send STANDBY, GO_2PC, and CANCEL messages. Controlled devices send STANDING_BY, COMPLETE, CANCELLED, and ABORT messages.

6.2. Human Operators

This specification assumes that a human operator normally is present at every 2PC controller and controlled device. A person serving in this capacity is called the "local operator." The local operator may simply monitor operations. Or, the local operator may perform safety verification functions, such as activating a deadman interlock switch. The local operator also may initiate a manual override of a controlled device as described in Section 6.4.5.

Presumably, the local operator effects his or her control via some type of console or other user interface. This interface is called the "local operator interface."

6.3. Relating Two-Phase Commit To Other MSC Messages

There are several important differences between Two-Phase Commit (2PC) and the other MIDI Show Control (MSC) messages. Understanding these differences is important to successfully building a product to use either message set. Also, the differences strongly suggest that any one controlled device should NOT support both message sets. So, understanding the differences is important to choosing the best message set to use in any given controlled device.

The MSC/2PC differences start with fundamental principles. MSC is dynamic and immediate: do this now, this way. 2PC is planned, scripted: do this sequence just like it was designed and no other way. MSC is well suited to rock and roll concerts, where there is no script and to media elements which have no safety implications, such as lighting, sound and video. 2PC is better suited to tightly scripted shows and to media elements which have safety implications, such as machinery, pyrotechnics and process control.

The MSC message set includes many messages that cause some kind of mechanical or electrical action by the controlled device, for example: GO, SET, FIRE, STOP, and RESUME. This large collection of action messages is required to communicate all the various desired results, with consistent behavior across a large variety of controlled devices. Appropriate behavior is ensured through explicit conformance to each action message defined in the MSC specification.

The 2PC message set contains just one action producing message, GO_2PC. Precisely what results are produced by any given GO_2PC message is determined by a script of cues stored in the controlled device,

and how the controlled device interprets that script of cues. Ensuring the desired behavior is a matter of selecting appropriate controlled devices and loading them with cue scripts that produce the correct results. Then, the controller must be programmed to issue 2PC messages in ways that properly coordinate the cue scripts in all the controlled devices.

The major advantage of MSC over 2PC is faster responses to requested actions and simpler system configuration. The faster response comes from two facts. First, MSC requires just one message to produce an action. 2PC requires at least three messages. Second, 2PC includes a delay, up to two seconds, between the request for an action and initiation of that action.

2PC has two advantages over MSC. First, 2PC can coordinate the actions of multiple controlled devices with an extremely high degree of certainty. Second, 2PC has error detection and recovery semantics built in to the protocol.

6.4. Two-Phase Commit Message Sequences

6.4.1. Normal Message Sequences

In the absence of any error conditions, four messages are required to execute a single cue in the two-phase commit protocol. The following table shows the messages, their ordering, the senders and receivers for each message, and the purpose of the message. The first and second messages comprise the 'standby' phase mentioned in Section 6. The third and fourth messages comprise the 'go' phase.

Order	Message	Sender	Purpose
1st	STANDBY	controller	to notify controlled device a cue is about to be executed (initiates a cue execution sequence)
2nd	STANDING_BY	controlled device	notify controller that the controlled device is ready and able to execute the cue described in a previous STANDBY message (also informs controller of time required to execute the cue)
3rd	GO_2PC	controller	instruct controlled device to begin execution of cue identified in previous STANDBY - STANDING_BY message pair
4th	COMPLETE	controlled device	notify controller that the controlled device has completed execution of the cue described in a previous GO_2PC message (terminates a cue execution sequence)

A controller may send STANDBY and GO_2PC messages to multiple controlled devices in any cue number order deemed appropriate by the controller. However, care must be taken with respect to the cue number order of messages sent to a single controlled device. Controlled devices may require a specific ordering of the cues that they execute. Failure to observe this ordering will result in the controlled device responding to one or more STANDBY messages with ABORT messages containing the "cue out of sequence" status code.

A controller must never make assumptions about the specific cue number order in which STANDING_BY, COMPLETE, or CANCELLED messages will be received from a controlled device. Such assumptions are clearly invalid for COMPLETE messages, since their order will depend on the time required to execute individual cues. Controlled devices are equally free to send STANDING_BY and CANCELLED messages in any order deemed appropriate, so long as the 2 second timeout interval rules discussed below are observed.

STANDBY messages should normally be sent at least 2 seconds before the anticipated time for sending the GO_2PC message. Otherwise, there may be insufficient time to discover that the controlled device has not responded with a STANDING_BY message within the 2 second timeout interval. It is recognized that sending STANDBY messages 2 seconds before GO_2PC messages may not always be possible. However, controller designers also must be aware of the risks associated with failure to observe the "2 second standby" rule, endeavor to observe the rule whenever possible, and be prepared for the consequences of failure to observe it.

6.4.2. Response Timeouts

The controller must record the cue execution time reported in each STANDING_BY message and timeout COMPLETE messages that are not received in 125% of that interval. The additional 25% allows for different timing mechanisms in the controller and controlled devices.

Timeout of response messages from a controlled device to the controller is a key element of the two-phase commit methodology. The timeout process requires that STANDING_BY and CANCELLED messages be returned within 2 seconds of delivery of the corresponding STANDBY or CANCEL message, and that COMPLETE messages be returned within the timeout interval described above.

Failure to meet timeout requirements signals the controller that the controlled device has become inoperative (at least in terms of its ability to participate in the two-phase commit protocol). Controllers that detect a timeout condition must treat the event as if an ABORT message had been delivered to the controller by the controlled device.

To simplify internal representation of the timeout event, a "timeout" status code is defined. Although no transmitted ABORT message will contain the "timeout" status code, controllers may represent a timeout condition using an internally generated and processed ABORT message that contains the "timeout" status code.

6.4.3. Exceptional Condition Handling

The ABORT, CANCEL, and CANCELLED two-phase commit messages are used only when exceptional conditions occur. The following table describes the purposes, senders and receivers for these messages.

Message	Sender	Purpose
ABORT	controlled device	notify controller of exceptional condition
CANCEL	controller	to instruct controlled device to discard previous instructions
CANCELLED	controlled device	confirms discarding of previous instructions

Whenever a controlled device detects an exceptional condition (something that prevents normal execution of a cue), it reports the condition to the controller using an ABORT message. In addition to a status code value that describes the exceptional condition, the ABORT message contains the sequence

number of the STANDBY or GO_2PC message that cannot be properly executed because of the exceptional condition.

Note: controlled devices cannot send ABORT messages except in response to STANDBY, GO_2PC, CANCEL messages. From the point of view of the larger presentation, the exceptional condition does not become significant until its existence prevents proper cue execution. However, controlled devices may still initiate local actions the instant the exceptional condition is detected.

In some cases, controllers may respond to ABORT messages by retransmitting the message that resulted in the ABORT message. This is discussed in Section 6.4.4. Otherwise, controllers respond to ABORT messages by displaying an informative message for their operators to read. The text of the operator message is based on the status code found in the ABORT message. In addition, CANCEL messages are sent for all relevant cues. Typically, CANCEL messages are sent for all cues for which STANDBY messages have been sent and all cues for which GO_2PC messages have been sent.

The general delivery of CANCEL messages informs all currently active controlled devices that an unusual condition exists and that show cue sequencing is about to enter a special recovery phase. This information is important. For example, controlled devices that do not usually allow out-of-sequence cue execution may allow it after receipt of a CANCEL message.

Sometimes, an ABORT message does not indicate the beginning of special recovery actions. For example, the ABORT message sent by the electric eye controlled device in Section 6.9.1 only indicates that an anticipated event has not occurred yet. In cases like this, the controller does not initiate general delivery of CANCEL messages upon receipt of an ABORT message.

After relevant activities are canceled, several courses of action are possible. The simplest possibility is that the human operators at the controlled devices must intervene to take whatever actions are possible to continue the performance. A more sophisticated controller might provide for contingency cue scripts that are activated either manually or automatically when an ABORT condition is reported.

The key aspects of the two-phase commit protocol involved in error handling are:

1. Provision of the ABORT message to signal the presence of an exceptional condition and something about the nature of that condition (via the status code),
2. Usage of the CANCEL message to inform all participating controlled devices of a deviation from the error-free cue execution sequence, and
3. The large set of predefined status code values. These definitions allow the controller to display an informative description of the problem to the controller operator, who is most likely responsible for recovering from the problem.

Controllers send CANCEL messages for cues that have been mentioned only in STANDBY messages and for cues that are already executing in response to GO_2PC messages. When the cue named in a CANCEL message has been mentioned only in a STANDBY message, the controlled device simply "forgets" that the STANDBY message was ever received. Things are much more complicated when a CANCEL message mentions a cue that is already executing.

Because cue execution is already in progress, the number of possible shutdown options grows dramatically. Picking a definitive right thing to do becomes much more difficult. Ultimately, the designers of two-phase commit controlled devices must consider carefully the choices between completing, pausing, terminating, and reversing cues whose execution has already been initiated. In addition, placing the cancel action choice in the hands of the controlled device operator should be considered.

If a currently executing cue is moving something out of harm's way, then completing that cue is probably the correct choice. If the currently executing cue is moving something into an uncertain situation, the cue probably should be paused, terminated, or reversed. Which of the three is best depends on the equipment and the performance situation specifics.

How cancellation of active cues is mechanically handled is the key safety component of the two-phase commit methodology. Since each situation is different, casting absolute requirements into this specification is impossible. All this specification can do is provide for the most complete set of options possible. Beyond that, controlled device designers are warned that special attention must be given to the cancellation of active cues.

6.4.4. Handling Exceptional Conditions With Message Retries

Under some circumstances, the controller may choose to retry transmission of a message that resulted in an exceptional condition, instead of requesting operator intervention. The most common case where retrying the message would be useful is an ABORT message with a "checksum error" status. Chances are good that the retransmitted message will arrive correctly. Other conditions may be recoverable via retransmission. Controller designers may choose to retry under any conditions that they think appropriate, and to retry as many times as seem useful. When retries are performed, however, they must be done as described in the remainder of this section.

Under no circumstances shall controlled devices use message retransmission as an exceptional condition recovery strategy.

Controllers shall retry for an ABORT message by retransmitting the message whose sequence number is found in the ABORT message. The controller may alter the message before retransmission, if logic indicates that such changes will improve chances for successful processing by the controlled device. For example, a controller may attempt retransmissions for ABORT messages with unknown <Q_list>" status codes. Part of this retransmission logic might involve dropping the <Q_list> datum from the retransmitted message.

Controllers also may use retransmission as a recovery from "timeout" errors or as part of treating incoming messages with checksum errors" as ABORT messages (Section 6.5). However, there is no way for the controller to ask the controlled device to retransmit the last message. Therefore, the controller must perform the retry by clearing and reestablishing the cue state in the controlled device.

The controller does this by sending CANCEL messages for every cue that the controlled device has acknowledged via a STANDING_BY message but for which no GO_2PC message has been sent. Then, the controller resends STANDBY messages for all the just canceled cues. If any of messages sent in this retransmission process also produces an error, the retransmission process must be considered to be a failure and operator intervention must be requested.

This "bad message from the controlled device" retry algorithm cannot attempt to clear cues for which GO_2PC messages have already been sent. To do so would mean stopping cue execution. That function must involve operator action.

6.4.5. Manual Override Processing

Controlled devices may be designed to ignore all two-phase commit MIDI show control messages whenever the local operator is manually initiating cue actions. (The "local operator" is described in Section 6.2.) When in effect, this condition is indicated by the "manual override in progress" status code. Controlled devices designed in this way must ignore ALL MSC_2PC messages. Selectively ignoring some messages but not others can cause system failures in the controller sending the messages.

In "manual override" mode, all STANDBY and GO_2PC messages must receive an ABORT message response containing the "manual override in progress" status code. All CANCEL messages must receive a CANCELLED message response containing the "manual override in progress" status code. This must continue until the local operator at the controlled device releases the manual override condition.

When the operator initiates a manual override during execution of a cue that was initiated by a GO_2PC message, an ABORT message containing the "manual override initiated" status should be sent at the time when the operator acts. Without delivery of the ABORT message, the controller may time out execution of the cue. The "manual override initiates" status indicates a change to "manual override" mode in the controlled device. This is different from the "manual override in progress" status, that indicates a continuation of an ongoing condition. The controller should appropriately notify its operator.

6.4.6. Waiting For Messages

The two-phase commit MIDI show control protocol is designed so that controlled devices are never waiting for messages. A controlled device simply accepts a message, executes the actions that the message requires, and returns a response message. A controlled device is required to "remember" the STANDBY - STANDING_BY messages exchanges that it has participated in recently, in order to process properly GO_2PC messages. Yet, strictly speaking, a controlled device is not stalled awaiting a GO_2PC message because it has previously received a STANDBY message.

One particular advantage of the no-waiting two-phase commit controlled device arrangement is worth noting. Since controlled devices are never waiting for messages, they are always prepared to accept manual override instructions from their local operator interfaces. Therefore, even if a controller should fail, the performance can continue using local manual operation of the individual controlled devices. (Manual override is described in Section 6.4.5.)

Although a controller by definition must wait for STANDING_BY, COMPLETE, and CANCELLED messages, two aspects of the waiting process suggest that the waiting cannot hang the controller (render it incapable of responding to inputs):

1. The amount of time that can elapse in such a wait is limited. The limit is 2 seconds for STANDING_BY and CANCELLED messages. The limit is variable (based on the contents of the previous STANDING_BY message) for COMPLETE messages.
2. A controller must be simultaneously waiting for numerous messages whose delivery order is indeterminate. Therefore, the controller must employ some form of table driven algorithm for tracking messages that are waiting for responses and detecting failures to receive responses within the specified limits.

6.5. Checksums

Each two-phase commit message includes a two byte (16 bits, in MIDI 14 data bits) checksum. To compute the checksum the <command_format> <command> and <data> portions of the message are treated as an array of two byte values. If there is an uneven number of bytes in the <command_format> <command> and <data> portions of the message, then an additional byte having the value zero is appended to the message for the checksum computation. Before the checksum computation, the byte positions that the checksum itself will occupy are zeroed.

A sum over the array of two byte values is computed. Overflows are ignored. Next, the <device_ID> byte is added to the sum. Finally, the sum is logically anded to the constant 7F 7Fh (32,639 decimal). This logical and operation makes the checksum value suitable for transport under MIDI. The resulting value is the checksum for the message.

The entity receiving a message will verify the correctness of each incoming message by recomputing the checksum and comparing it with the checksum value received. When a checksum comparison fails for a controlled device, it will return an ABORT message with a "checksum error" status code. When a checksum comparison fails for controller, it will proceed as if an ABORT message has been received. This may mean retrying the message transmission as described in Section 6.4.4.

6.6. Sequence Numbers

Sequence numbers are 14-bit (two MIDI byte) unsigned binary values. For each STANDBY, GO_2PC, and CANCEL message sent, a controller constructs a unique sequence number between 1 and 16,383. (The sequence number 0 is reserved for special functions and future protocol extensions.) A controlled device identifies the cue that its STANDING_BY, COMPLETE, CANCELLED, or ABORT message references by returning the sequence number received in the original STANDBY, GO_2PC, or CANCEL message.

Optionally, a controlled device may include explicit cue number information in addition to the sequence number in the STANDING_BY and COMPLETE messages. However, this information is only used as a sanity check on the message.

Using sequence numbers in this way permit inclusion of very simple controlled devices (such as safety interlock sensors) in two-phase commit operations. For example, a gas detector might have no concept of cues. Whenever a STANDBY message is received it simply checks for gas. It responds with a STANDING_BY message when gas is not detected, or an ABORT message when gas is detected. It ignores the <Q_number> etc. fields in the incoming message and simply copies the incoming sequence number into the response message.

Additionally, sequence numbers simplify response tracking operations in the controller.

6.7. Status Codes

Status codes appear in ABORT and CANCELLED messages. Status codes are two byte unsigned binary values. So that status codes may be transmitted in accordance with MIDI, the low-order two bits in a status code value must always be zero. The smallest legal status code value is 4, the largest legal value is FF FCh (65,532 decimal). In MIDI message descriptions, the status code appears as: s1 s2. Using unsigned integer arithmetic, the method for converting a status code to the s1 and s2 values, and vice versa is as follows:

```
s1 = (status_code /4) & 7F  
s2 = (status_code / 512) & 7F  
status_code = (s1 * 4) + (s2 * 512)
```

Each numeric status code value represents an error condition or canceled cue status. There are three numeric ranges of status codes. The first range is common to all command_format values. (See Section 4.1 for a discussion of command_format values.) Status codes in the first range apply to all types of MIDI show controllers. All status codes returned in CANCELLED messages fall into the first status code range.

The second status code range is additionally qualified by the command_format value appearing in the ABORT message. The exact meaning of these status codes depends on the type of device that sent it. For example, status code 10 08 means "water low" if received from a process control controlled device (command_format 50 through 5F). But, when received from a sound controlled device (command_format 10 through 1F), 10 08 means "amplifier failure."

The third status code range is qualified by both command_format and manufacturer.

The exact meaning of these status codes depends both on the type and manufacturer of the device that sent it. Manufacturers must publish information about all status codes in the third status code range used by their controlled devices.

The following table summarizes the status code ranges:

hex range	description
00 04 -- 0F FC	command_format & manufacturer dependent status codes (1023 possible values)
10 00 -- 7F FC	command_format dependent status codes (7,168 possible values)
80 00 -- FF FC	command_format independent status codes (8,192 possible values)
00 00	undefined status code (unknown error condition)

Note: command_format independent status codes can be easily detected because they are negative values when treated as signed values. Also, status code zero has been reserved to indicate an unknown error condition or an error condition for which no other status code value applies.

Manufacturers are free to use status codes in the manufacturer dependent range as they see fit. However, the highest degree of plug-and-play compatibility will be achieved if almost no status codes are in the manufacturer dependent range. Therefore, a simple and quick method will be provided for manufacturers to define relevant status code values in the command_format dependent range.

The tables below list all status codes that are independent of the command_format value. The letters in the messages column indicate which messages can produce a response containing the status code (S=STANDBY, G=GO_2PC, and C=CANCEL). The first table covers CANCELLED messages.

Status codes in CANCELLED messages

hex	messages	description
80 04	-- C	completing
80 08	-- C	paused
80 0C	-- C	terminated
80 10	-- C	reversed
80 24	-- C	not standing by
80 28	-- C	manual override in progress

The second status codes table covers ABORT messages. N.B. the only legal status codes in an ABORT message for a CANCEL message are "unknown/undefined error" and "checksum error." While the "timeout" status code will never appear in actual message transmission, it is included to simplify controller internal designs. All other cancel conditions are reported with a CANCELLED message.

Status codes in ABORT messages

hex	messages	description
00 00	S G C	unknown/undefined error
80 00	S G C	checksum error
80 20	s g c	*timeout
80 24	- G -	not standing by
80 28	S G -	manual override initiated
80 30	- G -	manual override in progress
80 40	S G -	deadman interlock not established
80 44	S G -	required safety interlock not established
80 50	S --	unknown <Q_number>
80 54	S --	unknown <Q_list>
80 58	S --	unknown <Q_path>
80 5C	S G -	too many cues active
80 60	S --	cue out of sequence
80 64	S G -	invalid d1 cue data value
80 68	S G -	invalid d2 cue data value
80 6C	S G -	invalid d3 cue data value
80 70	S G -	invalid d4 cue data value
80 90	S --	manual cueing of playback medium required
80 A0	S G -	power failure in controlled device subsystem
80 B0	S G -	reading new show cues from disk

* Used only internally by controllers. Should never appear in a transmitted abort message.

The presence of a status code does not require that it be used. Usage of a status code is required ONLY when specified in the detailed command and data descriptions in Section 5 or the general two-phase commit discussions in Sections 6 through 6.6. Status code values are listed here so that controlled device designers may have a broad range of values from which to choose when implementing this protocol. Also, this list provides a complete reference for the set of status code values that controller implementers should translate into useful, human readable text. For example, a controlled device may choose not to use the "cue out of sequence" status code at all. Or, a controlled device may choose to provide a method by which specific sets of cues must be executed in sequence. Such a controlled device would only return the "cue out of sequence" status code when its rules on sequential cueing are violated.

Special care must be taken in usage of the "manual override in progress" status code. Controlled devices may be designed to ignore all two-phase commit MIDI show control messages whenever the local console operator is manually initiating cue actions. When in effect, this condition is indicated by the "manual override in progress" status code. Controlled devices designed in this way must ignore ALL two-phase commit MIDI show control messages. Selectively ignoring some messages but not others can cause system failures in the controller sending the messages. See Section 6.4.5 for more details of manual override handling.

The following table lists status codes that are dependent on the command_format value but apply to all controller manufacturers. Letters in the messages column indicate which messages can produce a response containing the status code (S=STANDBY, and G=GO_2PC).

hex	messages	description
<i>For command_format values between 01 and 0F (lighting)</i>		
10 04	S G	position motor failure
10 08	S G	scroller motor failure
10 0C	S G	strobe not charged
10 10	S G	laser safety interlock not established
<i>For command_format values between 10 and 1F (sound)</i>		
10 04	S G	amplifier failure
10 08	S G	amplifier overload
<i>For command_format values between 20 and 2F (machinery)</i>		
10 04	S G	motor failure
10 08	S G	limit switch inhibiting movement
10 0C	S G	unequal movement in multiple section system
10 10	S G	servo failure
<i>For command_format values between 30 and 3F (video)</i>		
10 04	S G	sync lost
10 08	S G	time code lost
<i>For command_format values between 40 and 4F (projection)</i>		
10 04	S G	film tension lost
10 08	S G	lamp failure
<i>For command_format values between 50 and 5F (process control)</i>		
10 04	S G	hydraulic oil low
10 08	S G	water low
10 0C	S G	carbon dioxide low
10 10	S G	excess gas detected
10 14	S G	gas pilot out
10 18	S G	improper gas ignition conditions (windy)
10 1C	S G	smoke/fog fluid low
11 04	S G	invalid switch number
11 08	S G	latch setting system inoperative
12 04	S G	burned out cue light
<i>For command_format values between 60 and 6F (pyrotechnics)</i>		
10 04	S G	charge not loaded
10 08	S G	atmospheric conditions prohibit discharge

6.8. Cue Data Values (d1, d2, d3, and d4)

Both the STANDBY and the GO_2PC messages include 4 7-bit cue data values. These data values must be present in all STANDBY and GO_2PC messages, regardless of whether or not they are actually used by a controlled device. When correct d1 ... d4 values are unknown for a given cue or a given controlled device, zeros must be entered in the STANDBY or GO_2PC message in place of the d1 - d4 values.

The d1, d2, d3, and d4 data values allow controlled devices to rely on the controller to remember a small amount of information about how a cue is to be executed. Examples of how these values might be used appear later in this section. The d1 - d4 data values are not part of the cue numbering and identification scheme. Therefore, if a controlled device is expected to execute two cues with differing d1 ... d4 values simultaneously, the cues must be given different cue numbers.

The usage of the d1 - d4 values is defined by the designer of each controlled device. This definition must either:

- 1) conform to one of the common usage forms described in Sections 6.8.1 through 6.8.2, or
- 2) be clearly described in the documentation about the controlled device.

Option 1 is preferred. This specification will be updated as necessary in order to make the common usage forms appropriate for most controlled device manufacturers.

When a controlled device receives a d1, d2, d3, or d4 value that is incorrect, it must respond with an ABORT message containing one of the "invalid dn cue data value" status codes. Thus, an incorrect d1 value must result in an ABORT message containing the "invalid d1 cue data value" status code.

Those controlled device that do not use one or more of the d1, d2, d3, or d4 values shall not inspect the unused values for correctness. Suppose, for example, that a controlled device uses d2 and d3 (but not d1 and d4). That controlled device must check the correctness of all d2 and d3 values it receives. However, all values received in d1 and d4 must be ignored. Controlled devices that use none of the d1 through d4 values must ignore all of them.

In those cases where the error cannot be isolated to a single d1 - d4 value, the ABORT message must contain the status code that is appropriate for the lowest numbered data value involved. Suppose that a compound datum is constructed from the d3 and d4 values. When that compound datum is incorrect, an ABORT response message containing the "invalid d3 cue data value" status code must be sent.

6.8.1. Go-To-Level Lighting Console Usage of d1 and d2

Lighting consoles that operate on the "Go On, Go Off" concept use gl as a "Go Level" (where $gl = d1 + (d2 * 128)$). A STANDBY - GO_2PC sequence with $gl = 255$ is equivalent to Go On. A STANDBY - GO_2PC sequence with $gl = 0$ is equivalent to Go Off. gl values between 0 and 255 are also legal. They indicate that the specified cue should go to the level set by gl. For example, $gl = 128$ is equivalent to Go Cue To 50%. Any gl value not in the range 0 to 255 is currently illegal and must be responded to with an ABORT message containing an "invalid d1 cue data value" status code. The gl values 256 and above are reserved for possible future expansion of this capability.

6.8.2. Multiplexed Switch & Cue Light Usage of d1, d2, & d3

Multiplexed switch test and set boxes use sn as a switch number (where $sn = d1 + (d2 * 128)$). In addition, these boxes use d3 as a switch type value. The known switch type values are:

0	reserved
1	Close switch numbered sn
2	Open switch numbered sn
3	Test for switch number sn closed (aborts with "deadman interlock not established if open)
4	Test for switch number sn open (aborts with "deadman interlock not established" if closed)
10	Reset latch numbered sn
11	Test latch numbered sn (aborts with "deadman interlock not established" if not latched)
20	Operate cue light numbered sn

The electric eye used in the example in Section 6.9 could be tested using the numbered latch feature one of these switch boxes. However, the example assumes special, one of a kind, hardware.

6.9. Examples Of Two-Phase Commit Usage

The next several sections describe possible usage mechanics for the two-phase commit methodology. First, a basic, error free message exchange is described. Then, some example error conditions and how they might be handled are discussed.

This section is intended to provide guidance to someone implementing the two-phase commit protocol described previously in this document. It is not strictly a part of the protocol definition. The examples in this section are simply that: examples. Anyone who can devise a better way to implement something that conforms to the two-phase commit protocol described above is free to do so.

The examples in the following sections will be based on one coordinated cue. The cue involves a motorized turntable, which must rotate 180 degrees to expose the set that is upstage at the beginning of the cue. The complete turntable rotation through 180 degrees takes 30 seconds. Immediately down stage of the set on the turntable is a flown drop that must be raised by a motorized fly system. The turntable and fly system are both operated as MIDI controlled devices. They both have independent operator consoles.

Before the cue can begin an actress must exit the turntable. Unfortunately, this exit occurs in such a way that neither of the machinery operators can see it. Therefore, this system includes an electric eye arrangement designed so that the actress always breaks a light beam during the course of her exit. Before the fly or turntable can begin moving, the electric eye must have detected the actress' exit.

Of course, there are light and sound cues. There is one sound cue that begins when the actress breaks the electric eye beam and a second that begins when the turntable has turned half of its rotation (90 degrees). There is a light cue that begins with the fly starts to rise, another when the turntable starts to turn, and a final cue that begins when the turntable has completed its 180 degree rotation.

All these operations are coordinated by a controller MIDI show control computer. The functions are broken down into several component elements in a way that permits key events to be detected by the controller MIDI show control computer. For example, the turntable rotation is broken into two 90 degree rotations, instead of a single 180 degree rotation.

The following table contains a detailed cue list for the cues described above.

MIDI Two-Phase Commit Example Cue

Controller	Cue	Description
Electric Eye	EE-6	Actress breaks electric eye beam
Sound	S-109	First sound cue
Flys	F-28	Raise line 12 to 10 feet (clear set)
Lights	L-118	First light cue
Turntable	TT-34	Rotate 90 degrees
Lights	L-118.1	Second light cue
Flys	F-28.1	Raise line 12 to upper limit
Turntable	TT-34.1	Rotate 90 degrees
Sound	S-110	Second sound cue
Lights	L-119	Second light cue

The sequence in which these cues occur is:

1. Wait for the EE-6 to occur.
2. Go S-109.
3. Wait 3 seconds (to give the actress time to clear).
4. Go F-28 and L-118.
5. Wait for F-28 complete.
6. Go F-28.1, TT-34 and L-118.1.
7. Wait for TT-34 to complete.
8. Go TT-34.1 and S-110.
9. Wait for TT-34.1 to complete.
10. Go L-119.

The use of the L-119, TT-34.1 nomenclature is for the reader's convenience. The cue numbers transmitted in the MIDI messages would not include them. For example, MIDI message for TT-34.1 would have $<\text{command_format}>=24$ and $<\text{Q_number}>=34.1$ and L-119 would have $<\text{command_format}>=01$ and $<\text{Q_number}>=119$.

This sequence is conceptually based on an automated performance attraction at a major theme park. Some liberties have been taken so as to construct something that will show most the MIDI two-phase commit show control functions. If this sequence seems arbitrary or unrealistic, remember its principal use is as a basis for explaining MIDI show control two-phase commit, rather than an actual production situation.

6.9.1. Basic Message Exchanges

Now, let's consider how the basic cue sequence described above would be expressed in MIDI two-phase commit show control messages. This discussion will assume that no error conditions occur. The sequence of events is represented in the table below.

Time will flow from the top of the table to the bottom. The first column in the table is labeled seconds. The numbers in that column represent the approximate time the event occurs in seconds and thousandths of seconds.

Error-Free MIDI Two-Phase Commit Example Cue Execution

Seconds	Command	Format	Cue	Seq.	Status	Notes (see below)
00.000	STANDBY	10 (S)	109	001		A
00.001	STANDBY	22 (F)	28	002		
00.002	STANDBY	01 (L)	118	003		
00.003	STANDBY	24 (TT)	34	004		
00.004	STANDBY	01 (L)	118.1	005		
00.005	STANDBY	22 (F)	28.1	006		
00.010	STANDING_BY			001		B
00.011	STANDING_BY			002		
00.012	STANDING_BY			006		
00.013	STANDING_BY			003		
00.014	STANDING_BY			004		
00.015	STANDING_BY			005		
05.000	STANDBY	5F (EE)	6	007		
05.010	ABORT 007				80 40	C
05.500	STANDBY	5F (EE)	6	008		
05.510	ABORT 008				80 40	C
06.000	STANDBY	5F (EE)	6	009		
06.010	STANDING_BY			009		D
06.020	GO_2PC	10 (S)	109	010		
07.500	COMPLETE			010		
09.000	GO_2PC	22 (F)	28	011		E
09.001	GO_2PC	01 (L)	118	012		
09.500	GO_2PC	22 (F)	28.1	013		F
10.000	COMPLETE			012		
11.000	COMPLETE			011		
11.001	GO_2PC	24 (TT)	34	014		G
11.002	GO_2PC	01 (L)	118.1	015		
11.003	STANDBY	24 (TT)	34.1	016		
11.004	STANDBY	10 (S)	110	017		
11.005	STANDBY	01 (L)	119	018		
11.015	STANDING_BY			016		
11.016	STANDING_BY			017		
11.017	STANDING_BY			018		
16.000	COMPLETE			015		
21.000	COMPLETE			013		H
24.000	GO_2PC	24 (TT)	34.1	019		I
26.000	COMPLETE			014		J
26.001	GO_2PC	10 (S)	110	020		
34.000	COMPLETE			020		
41.000	COMPLETE			019		K
41.001	GO_2PC	01 (L)	119	021		
44.000	COMPLETE			021		

Notes:

- A. Send STANDBY messages for all cues that will go before the turntable completes 90 degrees of rotation. N.B. this must be done early enough to allow the full two second timeout interval to elapse before any actual cue execution is necessary.
- B. STANDING_BY response messages received. Only sequence numbers are used to identify the cues to which the STANDING_BY messages apply. Also, the STANDING_BY messages may not be received in the same order that the STANDBY messages were sent. In this case, the flys

controlled device gets its two STANDING_BY messages in back-to-back, even though other messages separated the STANDBY messages when they were sent.

- C. The electric eye STANDBY aborts because the actress has not broken the electric eye beam.
- D. The electric eye STANDING_BY message indicates that the actress has broken the electric eye beam. (Strict MIDI two-phase commit completeness requires that a CANCEL be sent for EE-6. But this is unnecessary since the controlled device does not require it.)
- E. Note the seconds column. There is an approximately three second delay between receipt of the STANDING_BY message at 06.010 and initiation of the flys and lights cues at 09.000.
- F. The flys cue 28.1 GO_2PC message is sent before the COMPLETE message is received for cue 28. The flys controller interprets this as an indication that line 12 should be kept moving while the COMPLETE message for cue 28 is sent.
- G. Once flys cue 28 reports complete, the turntable cue 34 and light cue 118.1 can be started. At this point, the standby message exchanges for the remaining cues also are performed. The same timeout considerations discussed in note A above apply here.
- H. Line 12 is completely flown at this point.
- I. The turntable cue 34.1 GO_2PC message is sent before the COMPLETE message is received for cue 34. Like the flys controlled device, the turntable controlled device interprets this as an indication that motion should be continued while the COMPLETE message for cue 34 is sent.
- J. Once turntable cue 34 reports complete, sound cue 110 can be started.
- K. Once turntable cue 34.1 reports complete, light cue 119 can be started.

6.9.2. Error Condition Detected Early

Now, some error conditions will be introduced into the basic cue sequence described in Section 6.9.1. First, consider an error that prevents execution of the entire cue sequence. Suppose the fly system control computer has detected a problem in one of the winch motors on line 12. It will return an ABORT message instead of a STANDING_BY message. The controller will respond to this by sending CANCEL messages for all cues that it previously sent STANDBY messages.

The message sequence would look something like:

Seconds	Command_	Format	Cue	Seq.	Status	Notes (see below)
00.000	STANDBY	10 (S)	109	001		
00.001	STANDBY	22 (F)	28	002		
00.002	STANDBY	01 (L)	118	003		
00.003	STANDBY	24 (TT)	34	004		
00.004	STANDBY	01 (L)	118.1	005		
00.005	STANDBY	22 (F)	28.1	006		
00.010	STANDING_BY			001		
00.011	ABORT			002	10 04	A
00.012	CANCEL	10 (S)	109	007		B
00.013	ABORT			006	10 04	
00.014	CANCEL	01 (L)	118	008		
00.015	STANDING_BY			003		
00.016	CANCEL	24 (TT)	34	009		
00.017	CANCEL	01 (L)	118.1	010		
00.018	CANCEL	22 (F)	28.1	011		
00.030	CANCELLED			007	80 0C	C
00.031	CANCELLED			009	80 0C	
00.032	CANCELLED			010	80 0C	
00.033	CANCELLED			008	80 0C	
00.034	CANCELLED			011	80 24	D

Notes:

- A. The first flys cue returns the ABORT message with a "motor failure" status. This status would be converted to text and displayed to the controller operator.
- B. CANCEL messages are sent to all controlled devices for all cues that have not yet completed execution. This conforms to the two-phase commit error recovery principles described in Section 6.4. Because the individual controlled devices are still processing the STANDBY messages, STANDING_BY and CANCEL messages are intermixed on the wire.
- C. The CANCELLED messages begin arriving. Like STANDING_BY messages, the CANCELLED messages need not be received in the same order that the CANCEL messages were sent. The CANCELLED status is "terminated" because none of the canceled cues ever began execution.
- D. The controller left nothing to chance. It sent a CANCEL message for F-28.1 even though an ABORT message for that cue was received at 00.013 seconds. Since the flys controlled device had already "forgotten" that F-28.1 was standing by, its CANCELLED message contains a "not standing by" status code.

The good news in this hypothetical situation is that the MIDI Show Control Two-Phase Commit protocol has detected and reported the inoperative fly system motor. Initiation of the turntable rotation has been automatically stopped. The set on the turntable will not rotate into and shred the drop. Last, but by no means least, the stage manager's MIDI show controller has alerted him/her to the situation. The bad news is that the fly system operator is going to have to work fast to deal with the bad winch motor on line 12.

6.9.3. Errors Detected During Execution Of A Cue

Error conditions that are detected during execution of one or more cues are dealt with in basically the same manner as is shown in the previous section. All pending or executing activities are ended using CANCEL messages. The major difference is that, because some actions are already in progress, the number of possible shutdown options grows dramatically. Picking a definitive right thing to do becomes much more difficult.

Ultimately, the designers of two-phase commit controlled devices must consider carefully the choices between completing, pausing, terminating, and reversing cues whose execution has already been initiated. For example, suppose that the motors driving the turntable in the basic cue sequence example fail after the turntable has rotated 45 degrees. The ABORT message would be sent at about 18.000 seconds in the table in Section 6.9.1.

CANCEL messages would be sent for the two cues that are standing by, S-110 and L-119. However, the interesting problem is handling the CANCEL message for F-28, which is not yet complete at that time. Since line 12 is going out, the safest thing to do is probably to complete F-28, returning a "completing" status in the CANCELLED message and ultimately a COMPLETE message. However, if line 12 were coming in, F-28 should not be completed. Pausing, terminating, or reversing the cue is the proper choice in that situation.

MIDI Machine Control 1.0

MMA0016 / RP013

Copyright © 1992 MIDI Manufacturers Association

ALL RIGHTS RESERVED. NO PART OF THIS DOCUMENT MAY BE REPRODUCED IN ANY FORM OR BY ANY MEANS, ELECTRONIC OR MECHANICAL, INCLUDING INFORMATION STORAGE AND RETRIEVAL SYSTEMS, WITHOUT PERMISSION IN WRITING FROM THE MIDI MANUFACTURERS ASSOCIATION.

MMA
POB 3173
La Habra CA 90632-3173

MIDI Machine Control Recommended Practice 1.00

January, 1992

CONTENTS

1	INTRODUCTION	1
	Numeric Conventions	1
2	GENERAL STRUCTURE	1
	Universal System Exclusive Format	1
	Open Loop vs. Closed Loop	2
	Handshaking	2
	Device Identification	3
	Commands	4
	Responses	4
	Extension Sets	4
	Data Lengths	5
	Segmentation	5
	Error Handling	6
	Command Message Syntax	6
	Response Message Syntax	7
3	STANDARD SPECIFICATIONS	8
	Standard Time Code	8
	Standard Short Time Code	9
	Standard User Bits	9
	Drop Frame Notes	10
	Standard Speed	11
	Standard Track Bitmap	11
	Motion Control States and Processes	12
	Motion Control State (MCS)	12
	Motion Control Process (MCP)	13
4	INDEX LIST	14
	Message Types	14
	Abbreviations Used	14
	Guideline Minimum Sets	14
	Commands	15
	Responses and Information Fields	16
	Commands and Information Fields according to Type	17
	Read and Write	17
	Transport Control	17
	Local Time Code	18
	Synchronization	18
	Generator	18

MIDI Time Code	18
Time Code Mathematics	19
Procedures	19
Event Triggers	19
Communications	19
5 DETAILED COMMAND DESCRIPTIONS	20
01 STOP	20
02 PLAY	20
03 DEFERRED PLAY	20
04 FAST FORWARD	21
05 REWIND	21
06 RECORD STROBE	21
07 RECORD EXIT	23
08 RECORD PAUSE	23
09 PAUSE	24
0A EJECT	24
0B CHASE	24
0C COMMAND ERROR RESET	25
0D MMC RESET	25
40 WRITE	26
41 MASKED WRITE	26
42 READ	26
43 UPDATE	27
44 LOCATE	28
45 VARIABLE PLAY	29
46 SEARCH	29
47 SHUTTLE	30
48 STEP	30
49 ASSIGN SYSTEM MASTER	30
4A GENERATOR COMMAND	31
4B MIDI TIME CODE COMMAND	31
4C MOVE	32
4D ADD	32
4E SUBTRACT	33
4F DROP FRAME ADJUST	33
50 PROCEDURE	34
51 EVENT	36
52 GROUP	39
53 COMMAND SEGMENT	40
54 DEFERRED VARIABLE PLAY	40
55 RECORD STROBE VARIABLE	41
7C WAIT	43
7F RESUME	43
6 DETAILED RESPONSE & INFORMATION FIELD DESCRIPTIONS	44
01 SELECTED TIME CODE	44
02 SELECTED MASTER CODE	44
03 REQUESTED OFFSET	45
04 ACTUAL OFFSET	45
05 LOCK DEVIATION	46
06 GENERATOR TIME CODE	46

07	MIDI TIME CODE INPUT	46
08	GP0 / LOCATE POINT	47
09	GP1	47
0A	GP2	47
0B	GP3	47
0C	GP4	47
0D	GP5	47
0E	GP6	47
0F	GP7	47
21	Short SELECTED TIME CODE	47
22	Short SELECTED MASTER CODE	47
23	Short REQUESTED OFFSET	47
24	Short ACTUAL OFFSET	47
25	Short LOCK DEVIATION	47
26	Short GENERATOR TIME CODE	47
27	Short MIDI TIME CODE INPUT	47
28	Short GP0 / LOCATE POINT	47
29	Short GP1	47
2A	Short GP2	47
2B	Short GP3	47
2C	Short GP4	47
2D	Short GP5	47
2E	Short GP6	47
2F	Short GP7	47
40	SIGNATURE	48
41	UPDATE RATE	50
42	RESPONSE ERROR	50
43	COMMAND ERROR	51
44	COMMAND ERROR LEVEL	54
45	TIME STANDARD	55
46	SELECTED TIME CODE SOURCE	55
47	SELECTED TIME CODE USERBITS	56
48	MOTION CONTROL TALLY	56
49	VELOCITY TALLY	58
4A	STOP MODE	58
4B	FAST MODE	59
4C	RECORD MODE	60
4D	RECORD STATUS	61
4E	TRACK RECORD STATUS	61
4F	TRACK RECORD READY	61
50	GLOBAL MONITOR	62
51	RECORD MONITOR	63
52	TRACK SYNC MONITOR	63
53	TRACK INPUT MONITOR	64
54	STEP LENGTH	65
55	PLAY SPEED REFERENCE	65
56	FIXED SPEED	65
57	LIFTER DEFEAT	66
58	CONTROL DISABLE	66
59	RESOLVED PLAY MODE	67
5A	CHASE MODE	67
5B	GENERATOR COMMAND TALLY	68
5C	GENERATOR SET UP	68
5D	GENERATOR USERBITS	69

5E	MIDI TIME CODE COMMAND TALLY	69
5F	MIDI TIME CODE SET UP	70
60	PROCEDURE RESPONSE	71
61	EVENT RESPONSE	71
62	TRACK MUTE	72
63	VITC INSERT ENABLE	72
64	RESPONSE SEGMENT	73
65	FAILURE	73
7C	WAIT	74
7F	RESUME	74
 Appendix A EXAMPLES		 75
	Example 1	75
	Example 2	76
	Example 3	82
 Appendix B TIME CODE STATUS IMPLEMENTATION TABLES		 90
01	SELECTED TIME CODE	90
02	SELECTED MASTER CODE	91
03	REQUESTED OFFSET	91
04	ACTUAL OFFSET	92
05	LOCK DEVIATION	92
06	GENERATOR TIME CODE	93
07	MIDI TIME CODE INPUT	94
08-0F	GPO thru GP7	95
 Appendix C SIGNATURE TABLE		 96
	Command Bitmap Array	96
	Response/Information Field Bitmap Array	97
 Appendix D MIDI MACHINE CONTROL and MTC CUEING		 99
	Comparison of MIDI Machine Control and MTC Cueing event specifications	99
	Review of MTC Cueing messages, and their relationship to MMC	100
 Appendix E DETERMINATION OF RECEIVE BUFFER SIZE		 102
	Operation of WAIT in a Simple "Closed Loop"	102
	External MIDI Mergers	104

1 INTRODUCTION

MIDI Machine Control is a general purpose protocol which initially allows MIDI systems to communicate with and to control some of the more traditional audio recording and production systems. Applications may range from a simple interface through which a single tape recorder can be instructed to PLAY, STOP, FAST FORWARD or REWIND, to complex communications with large, time code based and synchronized systems of audio and video recorders, digital recording systems and sequencers. Considerable expansion of the MIDI Machine Control protocol is realizable in the future, and many diverse audio, visual and mixed media devices may thus be brought together under a single general purpose control umbrella.

The set of Commands and Responses is modelled on the Audio Tape Recorder section of the Ebus standard. The intention is that command translation between the MIDI Machine Control specification and the Ebus standard will be relatively straight forward, being based on the same operating principles. On the other hand, it has been assumed that translation will involve more than table look-up, and considerable variation will be found in data specification and other communications details. In essence, MIDI Machine Control is intended to communicate easily with devices which are designed to execute the same set of operations as are defined in the Ebus standard.

By contrast with Ebus and other control protocols, MIDI Machine Control does not require that a Controlling device have intimate knowledge of the devices which it is controlling. In the simpler applications, a Controller will implement a set of commands deemed "reasonable" by its designers, and the Controlled Devices will apply their own intelligence to determine the best way to respond to commands received. At the same time, Controllers of much greater complexity can be designed around MIDI Machine Control, and applications are expected to extend from the very basic to the fully professional.

NUMERIC CONVENTIONS

All numeric quantities in this text should be assumed to be hexadecimal, unless otherwise noted.
All bit fields will be shown with the most significant bit first.

2 GENERAL STRUCTURE

UNIVERSAL SYSTEM EXCLUSIVE FORMAT

MIDI Machine Control uses two Universal Real Time System Exclusive ID numbers (sub-ID 1's), one for Commands (transmissions from Controller to Controlled Device), and one for Responses (transmissions from Controlled Device to Controller).

Throughout this document, "mcc" and "mcr" will be used to denote the Machine Control Command and Machine Control Response sub-ID 1's respectively. The resulting Real Time System Exclusives are as follows:

F0 7F <device_ID> <mcc> <commands . . . > F7

F0 7F <device_ID> <mcr> <responses . . . > F7

NOTES:

1. More than one command (or response) can be transmitted in a Sysex.
2. The number of bytes in a "commands" or "responses" field must not exceed 48.
3. Sysex's must always be closed with an F7 as soon as all currently prepared information has been transmitted.
4. Actual values for <mcc> and <mcr> are 06hex and 07hex respectively.

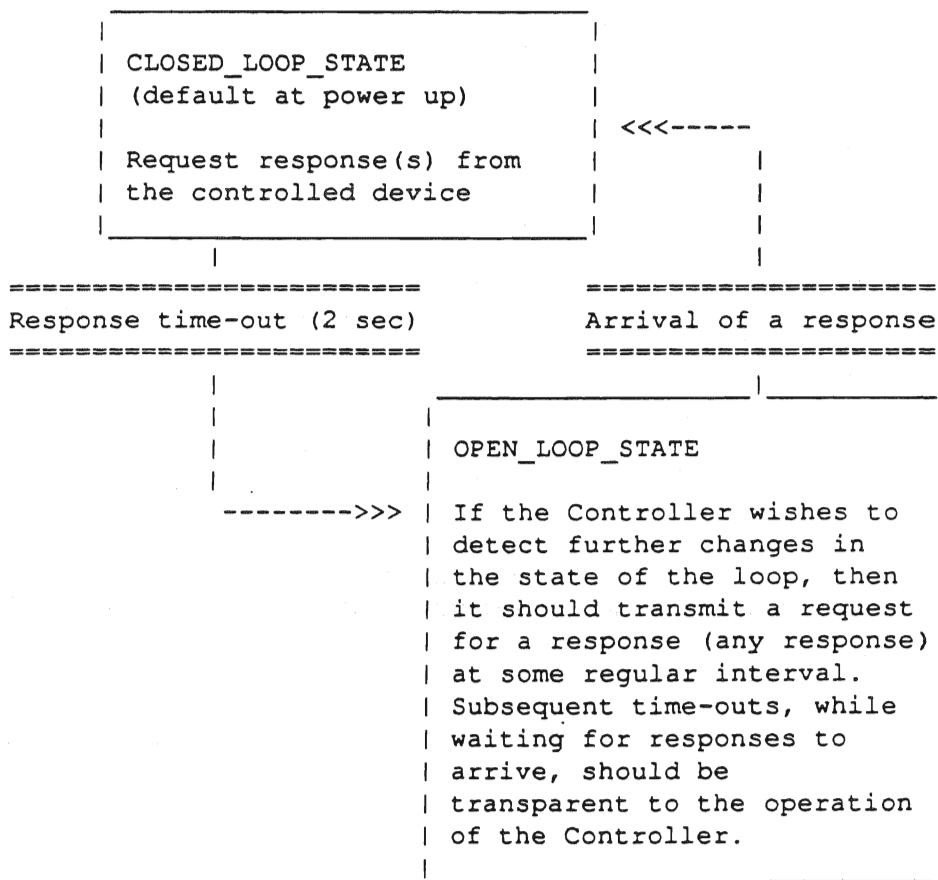
OPEN LOOP vs. CLOSED LOOP

MIDI Machine Control has been specifically designed for operation in both open and closed loop systems.

- Open Loop: a single cable connects the Controller's MIDI Out to the Controlled Device's MIDI In.
Closed Loop: an additional cable connects the Controlled Device's MIDI Out back to the Controller's MIDI In, allowing full duplex communication.

A Controller should power up expecting a closed loop. If, after issuing a command which expects a response, no response arrives within 2 seconds, then the loop can be assumed to be open.

Switching between these two states within the Controller may be represented as follows:



HANDSHAKING

Data flow is controlled, as required, by two simple messages:

- WAIT: "Please hold transmissions, my buffer is filling, etc"
and RESUME: "Please resume transmissions, my buffer is ready to receive again".

Each message must be transmitted as the only message in its particular System Exclusive. Handshaking from a Controller will be sent to the "all-call" address, while handshaking Responses from a Controlled Device will be identified by the device's own ID number. (See next section, "Device Identification".)

The four possible permutations of the WAIT and RESUME messages are:

From Controller to Controlled Device:

F0 7F <all_call=7F> <mcc> <WAIT> F7
F0 7F <all_call=7F> <mcc> <RESUME> F7

From Controlled Device to Controller:

F0 7F <device_ID> <mcr> <WAIT> F7
F0 7F <device_ID> <mcr> <RESUME> F7

Correct operation of the WAIT handshake requires a certain minimum size for the MIDI receive buffer in an MMC device. Refer to Appendix E, "Determination of Receive Buffer Size".

DEVICE IDENTIFICATION

Depending on context, <device_ID> is either a destination or a source address:

Commands: <device_ID> = DESTINATION device address
Responses: <device_ID> = SOURCE device address

Command strings are most often addressed to one device at a time. For example, to command two machines to play, transmit:

F0 7F <device_ID=machine 1> <mcc> <PLAY> F7
F0 7F <device_ID=machine 2> <mcc> <PLAY> F7

Group device_ID's are available for Commands. The previous example could be transmitted as:

F0 7F <device_ID=group 1> <mcc> <PLAY> F7, where "group 1" consists of machines 1 and 2.

The "all-call" device_ID (7F) may also be used for commands, and is useful for system wide "broadcasts".

Response strings, on the other hand, are always identified with a single device only.

The requirements (a) that a Controller be able to recognize a device_ID as a source, and (b) that a Controlled Device be prepared to recognize Group device_ID's, are unique to MIDI Machine Control, not being found in other Universal System Exclusive implementations.

Before fully interpreting the <device_ID> byte, parsing routines will need to look at <sub-ID#1>, which follows <device_ID>, in order to first determine that the Sysex contains Machine Control messages.

A typical system will consist of a Controller attached to one or more Controlled Devices. Responses from multiple Controlled Devices will have to be merged at some point, preferably within the Controller itself, using multiple MIDI IN's.

An external MIDI merging device is likely to work satisfactorily in most cases, but delays in the activation and delivery of the WAIT handshake may cause some problems where MIDI bandwidth is heavily utilized. (See also Appendix E "Determination of Receive Buffer Size".)

Although not recommended, it is possible that commands from more than one Controller could be merged and distributed to multiple Controlled Devices, with the device responses merged and fed back to the more than one Controllers. As all Controllers would be receiving all responses from all Controlled Devices, it is important that each Controller be prepared to receive device responses which were in fact requested by another Controller. Reliable error handling may have to be sacrificed when multiple Controllers are connected in this way. Some method should be provided so that error detection may be disabled at each Controller, assuming that error detection has been implemented in the first place. Refer to the COMMAND ERROR Information Field description for error handling details.

COMMANDS

Commands are messages from a Controller to a Controlled Device, or to a group of Controlled Devices. Each command has a command code between 01 and 77 hex, and may be followed by one or more data bytes. (Command 00 is reserved for extensions, and 78 thru 7F are reserved for handshaking.)

RESPONSES

Responses are messages from a Controlled Device to a Controller, and are usually transmitted in reaction to a command.

Conceptually, within the Controlled Device, data that is to be accessible to the Controller is maintained in an array of Information Fields (or internal "registers"). For example, the device's current time code may be found in the five byte Information Field called SELECTED TIME CODE; or the operating mode of its time code generator may be found in the three byte Field called GENERATOR SET UP.

Each Information Field has a name between 01 and 77 hex (00 is reserved for extensions, and 78 thru 7F are reserved for handshaking).

Most responses consist simply of an Information Field name followed by the data contained within that Field.

The READ and WRITE commands provide the Controller's primary access to the Controlled Device's Information Fields (each Information Field description indicates whether it is "read only" or "read/write"able).

For example, a READ command and response, where "<SELECTED TIME CODE>" represents the hexadecimal name of that Information Field:

Command: F0 7F <device_ID> <mcc> <READ> <count=01> <SELECTED TIME CODE> F7
Response: F0 7F <device_ID> <mcr> <SELECTED TIME CODE> hr mn sc fr st F7

A WRITE to the Information Field GENERATOR SET UP:

Command: F0 7F <device_ID> <mcc>
 <WRITE> <count=05>
 <GENERATOR SET UP> <count=03> <data1> <data2> <data3>
 F7
Response: none required

NOTE:

When an Information Field is listed as "read/write"able, then a READ will return data which reflects actual conditions at the device. This may or may not correspond to the data which was most recently written using the WRITE command.

EXTENSION SETS

Command 00 and Response/Information Field name 00 are reserved for two extension sets:

00 01 First Command or Information Field at first extension level.

00 00 01 First Command or Information Field at second extension level.

At this time, no extended functions have been defined. Nevertheless, to accommodate future extensions to MIDI Machine Control, parsing routines must always check for extensions wherever Command or Response/Information Field names are encountered in a data stream.

DATA LENGTHS

Since multiple Machine Control messages (i.e. Commands or Responses/Information Fields) may be transmitted within a single Sysex, and since the set of messages must be extendable in the future, it is necessary that any receiving device be able to determine the length of any received message, whether that message is known to the device or not.

Therefore, a large number of Commands and Information Fields include a byte count, while the remainder, in order to preserve bus bandwidth (particularly on the Response MIDI cable), have their length implied by the Hex value of their name byte.

COMMANDS:

00	reserved for extensions
01 thru 3F	0 data bytes
40 thru 77	Variable data, preceded by <count> byte
78 thru 7F	0 data ("handshake")

RESPONSES/INFORMATION FIELDS:

00	reserved for extensions
01 thru 1F	5 data bytes (standard time code fields)
20 thru 3F	2 data bytes ("short" time code fields)
40 thru 77	Variable data, preceded by <count> byte
78 thru 7F	0 data ("handshake")

NOTES:

1. Extension sets will follow this same format. For example, the Extended Response 00 27 would be followed by 2 data bytes.
2. Variable length data is of the form <count> <data . . . >, where <count> does not include the count byte itself.
3. It is possible that a variable length field could be extended in length in future versions of this specification, but currently defined contents will not be altered. For example, <count=04> <aa bb cc dd> may be extended, if required, to become <count=07> <aa bb cc dd xx yy zz>, but the definition of bytes <aa bb cc dd> will remain unchanged.
4. Variable length fields appear in three different formats in the text:
 - (i) pre-defined length e.g. <count=03> <pp qq rr>
 - (ii) length only partially defined due to the possible use of extension sets for Command or Information Field names within the data area e.g. <count=02+ext> <name1 name2>.
 - (iii) adjustable lengths e.g. <count=variable>.

SEGMENTATION

There will be some cases where a message (or string of messages) is too long to fit into a maximum length MMC System Exclusive data field (48 bytes). Such messages may be divided into segments and transmitted piece by piece across multiple System Exclusives.

Messages received in this way will be interpreted exactly as if they had arrived all in the same sysex.

Two specific messages support the segmentation process: COMMAND SEGMENT and RESPONSE SEGMENT. Each operates by embedding a segment of the larger message within its own data field. In addition, the first byte of that field contains a segment down counter, together with a flag (40hex) which marks the "first" segment. (The receiving device can therefore examine the first segment and ascertain how many more segments are to be transmitted.) The last segment will always have the down count byte set to zero.

For example, the command string *aa bb cc dd ee ff gg hh jj kk mm* (one large command or a string of smaller commands) would normally be transmitted as follows:

F0 7F <device_ID> <mcc> aa bb cc dd ee ff gg hh jj kk mm F7

Exactly the same result could be achieved using segmentation:

F0 7F <device_ID> <mcc> <COMMAND SEGMENT> <count=05> 42 aa bb cc dd F7

F0 7F <device_ID> <mcc> <COMMAND SEGMENT> <count=05> 01 ee ff gg hh F7

F0 7F <device_ID> <mcc> <COMMAND SEGMENT> <count=04> 00 jj kk mm F7

ERROR HANDLING

A RESPONSE ERROR message is transmitted from the Controlled Device to the Controller whenever a READ or UPDATE command requests data contained in an Information Field which is not supported by the device. In this way, the Controller will always receive at least one response for every data request that it makes, that is, it will always receive either the requested data or a RESPONSE ERROR message.

More details may be found in the descriptions of READ, UPDATE and RESPONSE ERROR.

Command errors, in contrast to response errors, are handled by the Information Fields COMMAND ERROR and COMMAND ERROR LEVEL, as well as the handshake message COMMAND ERROR RESET. All three of these messages must be implemented if command error handling is to be supported.

In its default state, a Controlled Device will attempt to ignore all command errors and to continue processing only those commands which arrive error-free. This method is most suited to "open loop" and other very basic systems. The more "intelligent" Controller can, however, enable either some or all of the defined command errors by writing to the COMMAND ERROR LEVEL field. Whenever an "enabled" error occurs, the Controlled Device halts all command processing and transmits the COMMAND ERROR message back to the Controller, giving details of the error and references to the command which caused it. The Controller must then issue a COMMAND ERROR RESET before normal operation can be resumed.

COMMAND MESSAGE SYNTAX

<command message> ::= F0 7F <destination> <mcc> <command string> F7

<destination> ::= <device address> | <group address> | <all call address>

<device address> ::= 00 | 01 | ... | 7E

<group address> ::= 00 | 01 | ... | 7E

<all call address> ::= 7F

<mcc> ::= 06 [sub-ID 1 for MIDI Machine Control Commands]

<command string> ::= <command> | <command string> <command>

<command> ::= <command_code_0>

| <command_code_variable> <count> <command data>
| <handshake>

<command_code_0> ::= 01 | 02 | ... | 3F | 00 <command_code_0>

<command_code_variable> ::= 40 | 41 | ... | 77 | 00 <command_code_variable>

<handshake> ::= 78 | 79 | ... | 7F | 00 <handshake>

RESPONSE MESSAGE SYNTAX

```
<response message> ::= F0 7F <source> <mcr> <response string> F7  
  
<source> ::= <device address>  
<device address> ::= 00 | 01 | ... | 7E  
  
<mcr> ::= 07 [sub-ID 1 for MIDI Machine Control Responses]  
  
<response string> ::= <response> | <response string> <response>  
  
<response> ::= <info_field_name_5> <standard 5-byte time code data>  
          | <info_field_name_2> <2-byte short time code data>  
          | <info_field_name_variable> <count> <info_field data>  
          | <handshake>  
  
<info_field_name_5> ::= 01 | 02 | ... | 1F | 00 <info_field_name_5>  
<info_field_name_2> ::= 20 | 21 | ... | 3F | 00 <info_field_name_2>  
<info_field_name_variable> ::= 40 | 41 | ... | 77 | 00 <info_field_name_variable>  
<handshake> ::= 78 | 79 | ... | 7F | 00 <handshake>
```

3 STANDARD SPECIFICATIONS

STANDARD TIME CODE (types {ff} and {st}):

This is the "full" form of the Time Code specification, and always contains exactly 5 bytes of data.

Two forms of Time Code subframe data are defined:

The first (labelled {ff}), contains subframe data exactly as described in the MIDI Cueing specification i.e. fractional frames measured in 1/100 frame units.

The second form (labelled {st}) substitutes time code "status" data in place of subframes. When processing time code data from a tape, for example, it is often useful to know whether "real" time code data is being received, or simply time data updated by the tape transport's tachometer pulses during a high speed wind.

Refer also to Appendix B "Time Code Status Implementation Tables" for exact usage of all the embedded status bits within each MMC time code Information Field.

hr mn sc fr {ff/st}

hr = Hours and type: 0 *tt hhhh*

tt = time type:

00 = 24 frame

01 = 25 frame

10 = 30 drop frame

11 = 30 frame

hhhh = hours (0-23 decimal, encoded as 00-17 hex)

mn = Minutes: 0 *c mmmmmmm*

c = color frame flag (copied from bit in time code stream):

0 = non color frame

1 = color framed code

mmmmmmmm = minutes (0-59 decimal, encoded as 00-3B hex)

sc = Seconds: 0 *k ssssss*

k = "blank" bit

0 = normal

1 = time code data has never been loaded into this Information Field

(i.e. since power up or an MMC RESET)

Set numeric time code value to all zeroes.

sssssss = seconds (0-59 decimal, encoded as 00-3B hex)

fr = Frames, byte 5 ident and sign: 0 *g i ffffff*

g = sign:

0 = positive

1 = negative (where signed time code is permitted)

i = final byte identification:

0 = subframes

1 = status

fffff = frames (0-29 decimal, encoded as 00-1D hex)

If final byte = subframes (*i* = 0):

ff = fractional frames: 0 *bbbbbbbb* (0-99 decimal, encoded as 00-63 hex)

If final byte = status (*i* = 1):

st = code status: 0 e v d n xxxx
e = estimated code flag:
 0 = normal time code
 1 = tach or control track updated code
v = invalid code (ignore if e=1):
 0 = this time code number has passed internal validation tests
 1 = validity of this time code number cannot be confirmed
d = video field 1 identification:
 0 = no field information in this frame
 1 = first frame in 4 or 8 field video sequence
n = "no time code" flag:
 0 = time code has been detected at the time code reader input
 1 = time code has never been read since power up or an MMC RESET
xxxx = reserved - must be set to 000.

STANDARD SHORT TIME CODE:

This shortened format may be used for repetitive response modes (see the UPDATE command), where the Controller instructs the Controlled Device to transmit data from a certain Information Field whenever it changes. The majority of such transmissions will contain some form of time code, and most of these will involve a change in only the frames portion when compared with the previous transmission. In other words, once an initial time code value has been transmitted, it is subsequently only necessary to transmit Hours, Minutes and Seconds data when a change takes place in any of them (i.e. once every second). The "Short" Time Code format therefore contains only Frames and Subframes data, and is identical to the frames and subframes portion of the "full" format:

fr {st/ff}

The major advantage of the "short" form is the preservation of response line bandwidth.

NOTES:

1. For every 5 byte time code Information Field name in the range 01h-1Fh, there is a corresponding 2 byte "short" time code field with its name in the range 21h-3Fh. For example, 06h is GENERATOR TIME CODE and 26h is Short GENERATOR TIME CODE.
2. The "short" forms are not individually described in the "Detailed Response and Information Field Descriptions" section. The format of each, however, may easily be deduced from the description of the corresponding "standard" form.

STANDARD USER BITS:

u1 u2 u3 u4 u5 u6 u7 u8 u9

u1 = Binary Group 1: 0000aaaa
u2 = Binary Group 2: 0000bbbb
u3 = Binary Group 3: 0000cccc
u4 = Binary Group 4: 0000dddd
u5 = Binary Group 5: 0000eeee
u6 = Binary Group 6: 0000ffff
u7 = Binary Group 7: 0000gggg

$u8$ = Binary Group 8: 0000hhhh
 $u9$ = Flags: 00000tji
 t = "secondary" time code bit
 0 = standard userbits
 1 = user bits contain "secondary" time code
 j = Binary Group Flag 1 (SMPTE time code bit 59; EBU bit 43)
 i = Binary Group Flag 0 (SMPTE time code bit 43; EBU bit 27)

NOTES:

1. Refer to the appropriate SMPTE and/or EBU standards for definition of the "Binary Groups" and of the "Binary Group Flags".
2. Time code may be occasionally encoded in userbits ($t = 1$). If so, the time code will be in the BCD form specified by SMPTE/EBU for normal time code (complete with the various SMPTE/EBU status flags as required), and loaded as follows:
 aaaa = Frames units
 bbbb = Frames tens
 cccc = Seconds units
 dddd = Seconds tens
 eeee = Minutes units
 ffff = Minutes tens
 gggg = Hours units
 hhhh = Hours tens
3. If the Binary Group nibbles 1-8 are used to carry 8-bit information, they should be reassembled as four 8-bit characters in the order hhhhhgggg fffffeeee ddddcrrrr bbbbaaaaa.
4. Display order for the userbits digits will also be hhhhhgggg fffffeeee ddddcrrrr bbbbaaaaa.

DROP FRAME NOTES

1. When writing to time code Information Fields, the drop-frame or non-drop-frame status of the data being written may be overridden by the status of the SELECTED TIME CODE.
 For example, if a tape recorder is reading drop-frame code from its tape, it will show drop-frame status in the SELECTED TIME CODE field. If a Controller subsequently loads the GP0/LOCATE POINT with a NON-drop-frame number and executes a LOCATE to GP0, then the GP0/LOCATE POINT will be interpreted as a drop-frame number, like SELECTED TIME CODE, with no attempt being made to perform any transformations.
2. Furthermore, if the above GP0/LOCATE POINT number had in fact been loaded with a non-existent drop-frame number (e.g. 00:22:00:00), then the next higher valid number would have been used (in this case, 00:22:00:02).
3. Calculation of Offsets, or simply the mathematical difference between two time codes, can cause confusion when one or both of the numbers is drop-frame.
 For the purposes of this specification, drop-frame numbers should first be converted to non-drop-frame before Offset calculations are performed. Results of the Offset calculation will then be expressed as non-drop-frame quantities.
 To convert from drop-frame to non-drop-frame, subtract the number of frames that have been "dropped" since the reference point 00:00:00:00. For example, to convert the drop-frame number 00:22:00:02 to non-drop-frame, subtract 40 frames, giving 00:21:58:22. The number 40 is produced by the fact that 2 frames were "dropped" at each of the minute marks 01 thru 09, 11 thru 19, 21 and 22.

STANDARD SPEED:

This three byte format is used by the VARIABLE PLAY, DEFERRED VARIABLE PLAY, RECORD STROBE VARIABLE, SEARCH and SHUTTLE Commands, as well as by the VELOCITY TALLY Information Field. It implies no specific resolution for speed control or velocity measurement internal to any Controlled Device.

sh sm sl

sh = Nominal Integer part of speed value: 0 *g sss ppp*

g = sign (1 = reverse)

sss = shift left count (see below)

ppp = most significant bits of integer multiple of play-speed

sm = MSB of nominal fractional part of speed value: 0 *qqqqqqqq*

sl = LSB of nominal fractional part of speed value: 0 *rrrrrrrr*

Speed values per shift left count:

<i>sss</i>	BINARY REPRESENTATION		USEABLE RANGES (DECIMAL)	
	Integer multiple of play-speed	Fractional part of play-speed	Integer range	Fractional resolution
000	<i>ppp</i>	. <i>qqqqqqqqrrrrrrrr</i>	0-7	1/16384
001	<i>pppq</i>	. <i>qqqqqqqr rr rr rr rr</i>	0-15	1/8192
010	<i>pppqq</i>	. <i>qqqqqqr rr rr rr rr</i>	0-31	1/4096
011	<i>pppqqq</i>	. <i>qqqqqr rr rr rr rr</i>	0-63	1/2048
100	<i>PPPqqqq</i>	. <i>qqqr rr rr rr rr</i>	0-127	1/1024
101	<i>PPPqqqqq</i>	. <i>qqrr rr rr rr rr</i>	0-255	1/512
110	<i>PPPqqqqqq</i>	. <i>qr rr rr rr rr rr</i>	0-511	1/256
111	<i>PPPqqqqqqq</i>	. <i>rr rr rr rr rr rr rr</i>	0-1023	1/128

STANDARD TRACK BITMAP:

This variable length field contains a single bit for each audio or video "track" supported by the Controlled Device. A bit value of 1 indicates an active state, while 0 indicates an inactive state. All unused or reserved bits must be reset to 0. The Standard Track Bitmap may be logically extended to 317 tracks before sysex segmentation is required.

The Standard Track Bitmap is currently used by the Information Fields TRACK RECORD READY, TRACK RECORD STATUS, TRACK SYNC MONITOR, TRACK INPUT MONITOR and TRACK MUTE.

When read as a Response, the Controlled Device need transmit only as many bytes of the Standard Track Bitmap as are required. Any track not included in a Response transmission will be assumed to be inactive, with its bit reset to zero. As the Standard Track Bitmap is always preceded by a byte count, a count of 00 may be used if all tracks are inactive.

When written to by a WRITE command, tracks not included in the transmission will have their individual bits reset to zero (track inactive). A Byte count of 00 may be used if all tracks are to be reset. Specific bits within the Standard Track Bitmap may also be modified by using the MASKED WRITE command.

r0 r1 r2 . . .

r0 Bitmap 0: 0 *gfedcba*
 a = Video
 b = reserved (must be zero)
 c = Time Code Track (dedicated)
 d = Aux Track A
 (e.g. analog guide tracks, etc.)
 e = Aux Track B
 f = Track 1 (stereo left / monaural)
 g = Track 2 (stereo right)
r1 Bitmap 1: 0 *nmlkjih*
 h = Track 3
 i = Track 4
 j = Track 5
 k = Track 6
 l = Track 7
 m = Track 8
 n = Track 9
r2 Bitmap 2: Tracks 10-16
r3 Bitmap 3: Tracks 17-23
r4 Bitmap 4: Tracks 24-30
r5 Bitmap 5: Tracks 31-37
r6 Bitmap 6: Tracks 38-44
r7 Bitmap 7: Tracks 45-51
r8 Bitmap 8: Tracks 52-58
r9 Bitmap 9: Tracks 59-65

.
.
.
etc

MOTION CONTROL STATES AND PROCESSES:

MOTION CONTROL STATE (MCS):

Basic transport commands such as PLAY, STOP, FAST FORWARD and REWIND will each move the Controlled Device to a new and mutually exclusive motion state. These commands are therefore collectively labelled as the "Motion Control State" commands. Each MCS command causes a transition into a new transport state and cancels the previous Motion Control State.

Receipt of a directly issued MCS command will also automatically terminate an active Motion Control Process (MCP), as described below (exceptions are the DEFERRED PLAY and DEFERRED VARIABLE PLAY commands when received during a LOCATE MCP).

MCS commands may be either:

- (i) directly issued by this command set,
- or (ii) indirectly issued as steps in the execution of a Motion Control Process (see below),
- or (iii) initiated elsewhere, for example, at the control panel of the device itself.

Motion Control State activity is tallied in the "Most recently activated Motion Control State" (*ms*) byte of the MOTION CONTROL TALLY Information Field. The device's success in achieving the requested state is tallied in the same field, in the "Status and success levels" (*ss*) byte.

All Motion Control State commands are marked "(MCS)" in the Index List and "(MCS command)" in the command descriptions.

Currently defined MCS commands are:

STOP
PAUSE
PLAY
DEFERRED PLAY
VARIABLE PLAY
DEFERRED VARIABLE PLAY
FAST FORWARD
REWIND
SEARCH
SHUTTLE
STEP
EJECT

MOTION CONTROL PROCESS (MCP):

Motion Control Processes (such as LOCATE and CHASE) are overriding control commands that cause the Controlled Device to automatically issue its own Motion Control State commands to achieve the desired result. Motion Control Processes are mutually exclusive and are commanded by MCP commands.

Receipt of an MCP command will override any previously received MCS command.

MCP commands may be either:

- (i) directly issued by this command set,
or (ii) initiated elsewhere, for example, at the control panel of the device itself.

Motion Control Process activities are tallied in the "Most recently activated Motion Control Process" (*mp*) byte of the MOTION CONTROL TALLY Information Field. The device's success in executing the requested process is tallied in the same field, in the "Status and success levels" (*ss*) byte.

In addition, during a Motion Control Process, each automatically activated Motion Control State will be registered in the MOTION CONTROL TALLY Information Field in the manner described in the previous section.

All Motion Control Process commands are marked "(MCP)" in the Index List and "(MCP command)" in the command descriptions.

Currently defined MCP commands are:

LOCATE
CHASE

4 INDEX LIST

MESSAGE TYPES

Each Command or Response/Information Field in the Index List has been assigned a Type designation as follows:

Comm	Support for communications e.g. WAIT, RESUME, GROUP.
Ctrl	Directly affects operation of the "transport" e.g. PLAY, STOP, TRACK RECORD READY, etc.
Evnt	Timed event triggering.
Gen	Time code generator interface.
I/O	READ and WRITE Commands, error handling, etc.
Sync	Used for time code synchronizing. Includes "master" time code fields.
Math	Time code mathematics.
MTC	MIDI Time Code input/output controls.
Proc	Definition and execution of PROCEDURE's (pre-defined command sequences).
Time	Information Fields directly related to the device's own time code stream.

ABBREVIATIONS USED

ATR	Audio Tape Recorder
{FF}	Time code contains "subframes" (see Section 3, Standard Specifications).
MCP	Motion Control Process *
MCS	Motion Control State *
MMC	MIDI Machine Control
r	Information Field is READ only.
RW	Information Field is READ/WRITE capable.
{st}	Time code contains "status" (see Section 3, Standard Specifications).
VTR	Video Tape Recorder

- * Each motion control command in the Index List is tagged "(MCS)" or "(MCP)", which indicates whether it will initiate a Motion Control State or a Motion Control Process, respectively. Refer to Section 3, "Standard Specifications", for an explanation of these terms.

GUIDELINE MINIMUM SETS

MIDI Machine Control does not specify an absolute minimum set of Commands and Responses/Information Fields which must be implemented in any device.

However, as an aid to understanding which commands and responses may be important in different situations, four Guideline Minimum Sets of Commands and Responses/Information Fields have been created:

- #1 Simple transport; no time code reader; "open loop" communications only.
- #2 Basic transport; no time code reader; "closed loop" communications possible.
- #3 Advanced transport; time code reader included; "closed loop" communications; event triggering functions; track by track record control.
- #4 Basic synchronizer; "closed loop" communications.

Guideline Minimum Sets are in no way intended to restrict the scope of operations of any device. They are offered only to help engineers trying to learn about MMC and perhaps looking to implement it for the first time. Assignment of any particular Command or Response/Information Field to a Guideline Minimum Set may be found in the far right hand column of the Index List.

Particular note should be taken of the SIGNATURE Information Field. This field contains a complete bit map of ALL Commands and Response/Information Fields supported by a Controlled Device. A Controller may, by interrogating the device's SIGNATURE, tailor its communications to exactly match the functions supported. Implementation of this SIGNATURE field is therefore highly recommended. A Controlled Device's signature should also be published by its manufacturer, using the format outlined in the SIGNATURE Information Field description.

COMMANDS

Hex	Command	Type	Number of data bytes	Guideline Min. Sets
00	reserved for extensions			1234
01	STOP (MCS)	Ctrl	-	1234
02	PLAY (MCS)	Ctrl	-	-234
03	DEFERRED PLAY (MCS)	Ctrl	-	1234
04	FAST FORWARD (MCS)	Ctrl	-	1234
05	REWIND (MCS)	Ctrl	-	1234
06	RECORD STROBE	Ctrl	-	1234
07	RECORD EXIT	Ctrl	-	1234
08	RECORD PAUSE	Ctrl	-	----
09	PAUSE (MCS)	Ctrl	-	----
0A	EJECT (MCS)	Ctrl	-	----
0B	CHASE (MCP)	Sync	-	---4
0C	COMMAND ERROR RESET	I/O	-	-234
0D	MMC RESET	Ctrl	-	1234
40	WRITE	I/O	n	1234
41	MASKED WRITE	I/O	n	--3-
42	READ	I/O	n	-234
43	UPDATE	I/O	n	-234
44	LOCATE (MCP)	Ctrl	n	1234
45	VARIABLE PLAY (MCS)	Ctrl	3	-234
46	SEARCH (MCS)	Ctrl	3	--34
47	SHUTTLE (MCS)	Ctrl	3	----
48	STEP (MCS)	Ctrl	1	----
49	ASSIGN SYSTEM MASTER	Sync	1	----
4A	GENERATOR COMMAND	Gen	1	----
4B	MIDI TIME CODE COMMAND	MTC	1	----
4C	MOVE	Math	2	1234
4D	ADD	Math	3	-234
4E	SUBTRACT	Math	3	-234
4F	DROP FRAME ADJUST	Math	1	--34
50	PROCEDURE	Proc	n	--34
51	EVENT	Evnt	n	--34
52	GROUP	Comm	n	-234
53	COMMAND SEGMENT	Comm	n	-234
54	DEFERRED VARIABLE PLAY (MCS)	Ctrl	3	-234
55	RECORD STROBE VARIABLE	Ctrl	3	----
7C	WAIT	Comm	-	-234
7F	RESUME	Comm	-	-234

RESPONSES AND INFORMATION FIELDS

Hex	Response/Information Field Name	Type	Number of data bytes	Read/ Write	Guideline Min. Sets
00	reserved for extensions				----
01	SELECTED TIME CODE {st}	Time	5	RW	1234
02	SELECTED MASTER CODE {st}	Sync	5	r	---4
03	REQUESTED OFFSET {ff}	Sync	5	RW	---4
04	ACTUAL OFFSET {ff}	Sync	5	r	---4
05	LOCK DEVIATION {ff}	Sync	5	r	---4
06	GENERATOR TIME CODE {st}	Gen	5	RW	----
07	MIDI TIME CODE INPUT {st}	MTC	5	r	----
08	GP0 / LOCATE POINT {ff}	Math	5	RW	1234
09	GP1 {ff}	Math	5	RW	-234
0A	GP2 {ff}	Math	5	RW	-234
0B	GP3 {ff}	Math	5	RW	-234
0C	GP4 {ff}	Math	5	RW	----
0D	GP5 {ff}	Math	5	RW	----
0E	GP6 {ff}	Math	5	RW	----
0F	GP7 {ff}	Math	5	RW	----
21 thru 2F	SHORT forms of 01 thru 0F		2	r	-234
40	SIGNATURE	I/O	n	r	-234
41	UPDATE RATE	I/O	1	RW	-234
42	RESPONSE ERROR	I/O	n	-	-234
43	COMMAND ERROR	I/O	n	r	-234
44	COMMAND ERROR LEVEL	I/O	1	RW	-234
45	TIME STANDARD	Time	1	RW	-234
46	SELECTED TIME CODE SOURCE	Time	1	RW	----
47	SELECTED TIME CODE USERBITS	Time	9	r	----
48	MOTION CONTROL TALLY	Ctrl	3	r	-234
49	VELOCITY TALLY	Ctrl	3	r	----
4A	STOP MODE	Ctrl	1	RW	----
4B	FAST MODE	Ctrl	1	RW	----
4C	RECORD MODE	Ctrl	1	RW	-234
4D	RECORD STATUS	Ctrl	1	r	-234
4E	TRACK RECORD STATUS	Ctrl	n	r	--3-
4F	TRACK RECORD READY	Ctrl	n	RW	--3-
50	GLOBAL MONITOR	Ctrl	1	RW	--3-
51	RECORD MONITOR	Ctrl	1	RW	----
52	TRACK SYNC MONITOR	Ctrl	n	RW	----
53	TRACK INPUT MONITOR	Ctrl	n	RW	----
54	STEP LENGTH	Ctrl	1	RW	----
55	PLAY SPEED REFERENCE	Ctrl	1	RW	-23-
56	FIXED SPEED	Ctrl	1	RW	----
57	LIFTER DEFEAT	Ctrl	1	RW	----
58	CONTROL DISABLE	Ctrl	1	RW	--4
59	RESOLVED PLAY MODE	Sync	1	RW	--4
5A	CHASE MODE	Sync	1	RW	--4
5B	GENERATOR COMMAND TALLY	Gen	2	r	----
5C	GENERATOR SET UP	Gen	3	RW	----
5D	GENERATOR USERBITS	Gen	9	RW	----

5E	MIDI TIME CODE COMMAND TALLY	MTC	2	r	----
5F	MIDI TIME CODE SET UP	MTC	1	RW	----
60	PROCEDURE RESPONSE	Proc	n	r	--34
61	EVENT RESPONSE	Evnt	n	r	--34
62	TRACK MUTE	Ctrl	n	RW	--3-
63	VITC INSERT ENABLE	Gen	3	RW	----
64	RESPONSE SEGMENT	Comm	n	-	-234
65	FAILURE	Ctrl	n	-	-234
7C	WAIT	Comm	-	-	-234
7F	RESUME	Comm	-	-	-234

COMMANDS AND INFORMATION FIELDS ACCORDING TO TYPE

READ and WRITE (I/O)

Commands:

- 0C COMMAND ERROR RESET
- 40 WRITE
- 41 MASKED WRITE
- 42 READ
- 43 UPDATE

Information Fields:

- 40 SIGNATURE
- 41 UPDATE RATE
- 42 RESPONSE ERROR
- 43 COMMAND ERROR
- 44 COMMAND ERROR LEVEL

TRANSPORT CONTROL (Ctrl)

Commands:

- 01 STOP (MCS)
- 02 PLAY (MCS)
- 03 DEFERRED PLAY (MCS)
- 04 FAST FORWARD (MCS)
- 05 REWIND (MCS)
- 06 RECORD STROBE
- 07 RECORD EXIT
- 08 RECORD PAUSE
- 09 PAUSE (MCS)
- 0A EJECT (MCS)
- 0D MMC RESET
- 44 LOCATE (MCP)
- 45 VARIABLE PLAY (MCS)
- 46 SEARCH (MCS)
- 47 SHUTTLE (MCS)
- 48 STEP (MCS)
- 54 DEFERRED VARIABLE PLAY (MCS)
- 55 RECORD STROBE VARIABLE

Information Fields:

- 48 MOTION CONTROL TALLY
- 49 VELOCITY TALLY

4A	STOP MODE
4B	FAST MODE
4C	RECORD MODE
4D	RECORD STATUS
4E	TRACK RECORD STATUS
4F	TRACK RECORD READY
50	GLOBAL MONITOR
51	RECORD MONITOR
52	TRACK SYNC MONITOR
53	TRACK INPUT MONITOR
54	STEP LENGTH
55	PLAY SPEED REFERENCE
56	FIXED SPEED
57	LIFTER DEFEAT
58	CONTROL DISABLE
62	TRACK MUTE
65	FAILURE

LOCAL TIME CODE (Time)

Information Fields:

01	SELECTED TIME CODE {st}
21	Short SELECTED TIME CODE {st}
45	TIME STANDARD
46	SELECTED TIME CODE SOURCE
47	SELECTED TIME CODE USERBITS

SYNCHRONIZATION (Sync)

Commands:

0B	CHASE (MCP)
49	ASSIGN SYSTEM MASTER

Information Fields:

02	SELECTED MASTER CODE {st}
03	REQUESTED OFFSET {ff}
04	ACTUAL OFFSET {ff}
05	LOCK DEVIATION {ff}
22	Short SELECTED MASTER CODE {st}
23	Short REQUESTED OFFSET {ff}
24	Short ACTUAL OFFSET {ff}
25	Short LOCK DEVIATION {ff}
59	RESOLVED PLAY MODE
5A	CHASE MODE

GENERATOR (Gen)

Command:

4A	GENERATOR COMMAND
----	-------------------

Information Fields:

06	GENERATOR TIME CODE {st}
26	Short GENERATOR TIME CODE {st}
5B	GENERATOR COMMAND TALLY
5C	GENERATOR SET UP
5D	GENERATOR USERBITS
63	VITC INSERT ENABLE

MIDI TIME CODE (MTC)

Command:

4B MIDI TIME CODE COMMAND

Information Fields:

07 MIDI TIME CODE INPUT {st}
27 Short MIDI TIME CODE INPUT {st}
5E MIDI TIME CODE COMMAND TALLY
5F MIDI TIME CODE SET UP

TIME CODE MATHEMATICS (Math)

Commands:

4C MOVE
4D ADD
4E SUBTRACT
4F DROP FRAME ADJUST

Information Fields:

08 GP0 / LOCATE POINT {ff}
09 GP1 {ff}
0A GP2 {ff}
0B GP3 {ff}
0C GP4 {ff}
0D GP5 {ff}
0E GP6 {ff}
0F GP7 {ff}
28 Short GP0 / LOCATE POINT {ff}
29 Short GP1 {ff}
2A Short GP2 {ff}
2B Short GP3 {ff}
2C Short GP4 {ff}
2D Short GP5 {ff}
2E Short GP6 {ff}
2F Short GP7 {ff}

PROCEDURES (Proc)

Command:

50 PROCEDURE

Information Field:

60 PROCEDURE RESPONSE

EVENT TRIGGERS (Evt)

Command:

51 EVENT

Information Field:

61 EVENT RESPONSE

COMMUNICATIONS (Comm)

Commands:

52 GROUP
53 COMMAND SEGMENT
7C WAIT
7F RESUME

Information Fields:

64 RESPONSE SEGMENT
7C WAIT
7F RESUME

5 DETAILED COMMAND DESCRIPTIONS

Messages from the CONTROLLER to the CONTROLLED DEVICE.

00 Reserved for extensions

01 STOP (MCS command)

Stop as soon as possible.

Output monitoring is controlled by the STOP MODE Information Field, if supported.

STOP will be cancelled by the receipt of another MCS or MCP command.

Recording [rehearsing] tracks exit from record [rehearse].

01 STOP

02 PLAY (MCS command)

Enter playback mode.

PLAY will be cancelled by the receipt of another MCS or MCP command.

02 PLAY

NOTE:

Recording [rehearsing] tracks do not automatically exit from record [rehearse] upon receipt of the PLAY command. If that action is desired, then transmit <RECORD EXIT> <PLAY>.

03 DEFERRED PLAY (MCS command)

Identical to the PLAY command, with the exception that if the device is currently executing a LOCATE (MCP), then PLAY mode will not be invoked until the LOCATE is completed.

Receipt of any other MCS or MCP command will cancel DEFERRED PLAY.

When received while a LOCATE is in progress, the "MCP Success Level" field of the MOTION CONTROL TALLY Information Field must be set to indicate "Actively locating with Deferred Play pending" for the duration of the LOCATE.

When the LOCATE has concluded:

- (i) An automatic PLAY (MCS) command will be issued;
- (ii) The MOTION CONTROL TALLY "MCS command" byte will switch to "PLAY";
- (iii) The MOTION CONTROL TALLY "MCP command" byte will become "No MCP's currently active", clearing both the LOCATE and Deferred Play indications.

If the device is not executing or does not support the LOCATE command, then it should immediately enter playback mode.

03 DEFERRED PLAY

NOTES:

1. If a device is to support only a single Play command, then DEFERRED PLAY is recommended. In the open loop case, it will not be possible for a Controller to simulate the operation of this command, as no "locate complete" status will be available. On the other hand, an immediate PLAY may be re-created by issuing STOP followed by DEFERRED PLAY.
2. Recording [rehearsing] tracks do not automatically exit from record [rehearse] upon receipt of the DEFERRED PLAY command. If that action is desired, then transmit <RECORD EXIT> <DEFERRED PLAY>.

04 FAST FORWARD (MCS command)

Move forward at maximum possible speed.

Output monitoring is controlled by the FAST MODE Information Field, if supported.

FAST FORWARD will be cancelled by the receipt of another MCS or MCP command.

Recording [rehearsing] tracks exit from record [rehearse].

04

FAST FORWARD

05 REWIND (MCS command)

Move in reverse direction at maximum possible speed.

Output monitoring is controlled by the FAST MODE Information Field, if supported.

REWIND will be cancelled by the receipt of another MCS or MCP command.

Recording [rehearsing] tracks exit from record [rehearse].

05

REWIND

06 RECORD STROBE

Switches the Controlled Device into or out of record or rehearse, according to the setting of the RECORD MODE Information Field. RECORD STROBE will be honored under two conditions only:

CONDITION 1: Controlled Device already Playing:

If the Controlled Device is already playing (i.e. the "Most recently activated Motion Control State" in the MOTION CONTROL TALLY Information Field is PLAY or VARIABLE PLAY), then RECORD STROBE will cause record [rehearse] entry on all tracks which are presently in a record ready state, and cause record [rehearse] exit on any currently recording [rehearsing] tracks which are no longer record ready. *1*8*9

CONDITION 2: Controlled Device Stopped:

If, when RECORD STROBE is received, the Controlled Device is completely stopped as a result of an explicit STOP or PAUSE command (i.e. [i] the "Most recently activated Motion Control State" in the MOTION CONTROL TALLY Information Field is STOP or PAUSE; [ii] the "MCS success level" shows "Completely stopped"; and [iii] the "Most recently activated Motion Control Process" byte is set to "No MCP's currently active"), then:

- (i) An automatic PLAY (MCS) command will be issued; *3*4
- (ii) At an appropriate point in the start up phase of the device, record [rehearse] entry will occur on all tracks which are presently in a record ready state. *1*5

06

RECORD STROBE

NOTES:

- *1. Tracks are switched in and out of the record ready state using the TRACK RECORD READY Information Field.
- 2. It is recommended that, for new Controlled Device designs based on MMC, no recording [rehearsing] should take place following a RECORD STROBE command unless at least one track is in a record ready state.
Among existing non-MMC devices, however, it is quite common that, given that record [rehearse] has been enabled (for example by setting the appropriate value in the RECORD MODE Information Field), recording [rehearsing] will be initiated even if no tracks are in a record ready state when the command to record [rehearse] is received. Such operation remains permissible under MMC, provided that the resultant status is correctly indicated by including the "No Tracks Active" bit in the RECORD STATUS Information Field.
- *3. Under CONDITION 2, an automatic PLAY (MCS) command will be issued only under the STOP or PAUSE conditions specified. At no other time does RECORD STROBE have any implications regarding play mode or playing speed.
- *4. Also under CONDITION 2, the automatic PLAY (MCS) command will be issued whether or not any tracks are in a record ready state, and whether or not a RECORD MODE has been specified.
- *5. The existence of a "record-pause" condition will not affect the operation of RECORD STROBE.
(Refer to the RECORD PAUSE and PAUSE command descriptions.)
- 6. Devices which support and have their RECORD MODE set to "VTR:Insert" or "VTR:Assemble", are expected to produce clean and correctly timed transitions into record [rehearse] under both Conditions 1 and 2 outlined above. When starting from STOP or PAUSE mode (Condition 2), it will usually be necessary to wait until the device's start up phase has been completed before recording [rehearsing] is attempted.
- 7. A Controlled Device will ignore any RECORD STROBE command which is received while it is neither already in play mode nor completely stopped as described.
- *8. Under CONDITION 1, "Controlled Device already Playing", it is not necessary for the PLAY or VARIABLE PLAY command to have been "successful" before RECORD STROBE is accepted. If, however, the desired play motion has not yet been achieved when RECORD STROBE is received, it may be necessary for the device to defer the onset of recording until an appropriate point in its start up phase.
- *9. Note also that, under CONDITION 1, recording is not inhibited by Motion Control Process activity.

07 RECORD EXIT

Causes a record exit on all currently recording tracks.

07

RECORD EXIT

08 RECORD PAUSE

Causes the Controlled Device to enter "record-pause" mode provided that a PAUSE mode is already in effect (i.e. that the most recent Motion Control State is PAUSE). *4

No actual recording is initiated by RECORD PAUSE, but the device is placed in a state from which a transition into record mode can be made quickly and smoothly.

All "record ready" track Outputs are switched to monitor their respective Inputs.

A Controlled Device which supports the RECORD PAUSE command must fully implement the "record-pause" mode as described here and under the PAUSE command itself.

A Controlled Device which does not support this command must never enter "record-pause" mode.

Once in "record-pause", the effect of further commands is as follows:

PAUSE:	No change
RECORD PAUSE:	No change
RECORD EXIT:	Cancellation of "record-pause", return to PAUSE
RECORD STROBE:	Smooth resumption of Play and Record. *6
RECORD STROBE VARIABLE:	Smooth resumption of Variable Speed Play and Record. *6
Any other MCS or MCP command:	Exit "record-pause" mode prior to further action. *7

"Record-pause" is tallied in the RECORD STATUS Information Field.

08

RECORD PAUSE

NOTES:

1. RECORD PAUSE is the only way to enter "record-pause" from a non-record PAUSE without initiating motion.
2. RECORD PAUSE in itself will not produce a PAUSE.
3. The command sequence <PAUSE> <RECORD PAUSE> will always produce a paused state, and, if implemented, the "record-pause" condition.
- *4. It is not necessary for the PAUSE command to have been "successful" in order for a RECORD PAUSE command to be accepted. Therefore, after receiving a <PAUSE> <RECORD PAUSE> command sequence, it may be necessary for the Controlled Device to defer the onset of "record-pause" until all motion has ceased following the PAUSE command. RECORD PAUSE must not cause any actual recording to take place.
5. RECORD PAUSE will be ignored by a Controlled Device if its Motion Control State is not already "PAUSE".
- *6. Refer also to the RECORD STROBE and RECORD STROBE VARIABLE command definitions.
- *7. This category includes PLAY, DEFERRED PLAY, VARIABLE PLAY and DEFERRED VARIABLE PLAY, none of which will cause recording to take place following a RECORD PAUSE.
8. No "rehearse-pause" mode is currently supported.

09 PAUSE (MCS command)

Stop as soon as possible.

VTR's and other visual devices stop "with picture".

The transport mechanism should be left in a state such that start up time will be minimized.

PAUSE will be cancelled by the receipt of another MCS or MCP command.

Recording tracks exit from record, with one exception: if (and only if) a device supports the RECORD PAUSE command, and if the PAUSE command is received while recording is taking place, then the "record-pause" condition will be invoked. (Please refer to the RECORD PAUSE command description). Rehearsing tracks always exit from rehearse.

09

PAUSE

NOTES:

1. Repetition of the PAUSE command will not produce the pause/play toggling action found on some cassette type VTR's.
2. To prevent head clogging, VTR's may switch to a "without picture" state after a pre-determined time-out period. A subsequent PAUSE command will cause a return of the picture.
3. The PAUSE command implies an automatic "standby on" or "ready" command for devices which have this capability.
4. Devices which do not normally support a separate pause function may still implement the PAUSE Command by substituting a STOP. However, MOTION CONTROL TALLY must indicate PAUSE.

0A EJECT (MCS command)

Removeable media devices eject media (cassette or disk etc.) from the transport mechanism.

When used as a tally in the MOTION CONTROL TALLY Information Field, EJECT indicates a lack of media which is irrecoverable without operator intervention.

Recording [rehearsing] tracks exit from record [rehearse].

0A

EJECT

NOTE:

EJECT may be used as a tally by devices which do not normally support EJECT as a command. For example, a reel-to-reel device should show an EJECT tally when its tape has run off the end of the reel.

0B CHASE (MCP command)

Causes the Controlled Device to attempt to follow, establish and maintain synchronism with the SELECTED MASTER CODE. When the SELECTED MASTER CODE is detected to be in "play" mode, the Controlled Device should play and attempt to synchronize. At other times, the Controlled Device should simply attempt to position itself so that, should the SELECTED MASTER CODE enter "play" mode, synchronism can be achieved in the minimum possible time.

Synchronism is to be achieved between the Controlled Device's SELECTED TIME CODE and the SELECTED MASTER CODE with an offset specified by the REQUESTED OFFSET Information Field using the formula:

$$\text{REQUESTED OFFSET} = \text{SELECTED TIME CODE} - \text{SELECTED MASTER CODE}.$$

If, when the CHASE command is received, the "blank" bit in the REQUESTED OFFSET Information Field is set (i.e. a time code value has never been loaded into it), then a "Blank time code" COMMAND ERROR will be generated, and Chase status will be set to "Failure".

The same error will be generated if the Controlled Device is unable, through its own actions, to eliminate a "blank" bit in either SELECTED TIME CODE or SELECTED MASTER CODE.

The CHASE MODE Information Field defines the type of synchronization which is to take place.

CHASE will be cancelled by the receipt of another MCP or MCS command.

0B

CHASE

0C

COMMAND ERROR RESET

Resets the "Error halt" flag in the COMMAND ERROR Information Field, allowing resumption of command processing in the Controlled Device. Refer to the COMMAND ERROR and COMMAND ERROR LEVEL Information Field descriptions.

0C

COMMAND ERROR RESET

0D

MMC RESET

Resets the Controlled Device's MIDI Machine Control communications channel to its power up condition, plus:

- (a) empties the UPDATE list;
- (b) deletes all PROCEDURE's;
- (c) deletes all EVENT's;
- (d) clears all GROUP assignments;
- (e) clears the COMMAND ERROR field, including the "Error halt" flag;
- (f) resets COMMAND ERROR LEVEL to zero ("All errors disabled");
- (g) sets all time code Information Field flags to their default state,
as outlined in Appendix B;
- (h) resets the MCP command tally byte in the MOTION CONTROL TALLY Information Field to 7Fhex ("No MCP's currently active");
- (i) cancels any command sysex de-segmentation which may be in progress.

MMC RESET must be supported by all MMC Controlled Devices.

0D

MMC RESET

40

WRITE

Loads data into the specified Information Field(s).

The WRITE command is followed by one or more strings each consisting of an Information Field name together with the complete data which is to be loaded into that Information Field.

Information Field data formats must be exactly as defined in "Detailed Response and Information Field Descriptions".

The specified Information Field must be writeable.

40	WRITE
<count=variable>	Byte count of following information
<name>	Writeable Information Field name
<data..>	Format defined by the Information Field name.
<name>	More <name> <data..> pairs as required . .
<data..>	

41

MASKED WRITE

Allows specific bits to be altered in a bitmap style Information Field (see Note *1).

To be "mask-writeable", the Information Field must be in the <count> <data> format, and the bitmap must begin immediately after the <count> byte.

41	MASKED WRITE
<count=04+ext>	Byte count of following data.
<name>	Mask-writeable Information Field Name *1
<byte #>	Byte number of target byte within bitmap. Byte 0 is the first byte after the <count> field in the mask-writeable Information Field definition.
<mask>	One's in this mask indicate which bits will be changed in the target bitmap byte.
<data>	New data for the target bitmap byte.

NOTES:

- *1. The only mask-writeable Information Fields currently defined are those which employ the Standard Track Bitmap (see Section 3 "Standard Specifications").
- 2. The "all one's" value for both <mask> and <data> is, of course, 7F.

42

READ

Requests transmission of the contents of the specified Information Field(s).

If an Information Field is unsupported by the Controlled Device, then a RESPONSE ERROR message will be generated. (Refer to the RESPONSE ERROR Information Field description.)

42	READ
<count=variable>	Byte count (not including command and count).
<name>	Information Field name(s)

FORMAT 1 - UPDATE [BEGIN]

UPDATE [BEGIN] causes a Controlled Device to immediately:

- (i) transmit the contents of the specified Information Field(s);
- and (ii) add the specified Information Field name(s) to an internal UPDATE "list". *1

Following this, at intervals no more frequent than that specified by the UPDATE RATE Information Field, the Controlled Device will:

- (iii) re-transmit the contents of any of the Information Fields in the internal UPDATE "list" if those contents have changed and have become different to the contents most recently transmitted as a result of the UPDATE command. *2*3

If any of the specified Information Fields are unsupported by the Controlled Device (or are undefined, or are defined as having "no access"), then a RESPONSE ERROR message will be generated naming the field(s) in error. (Valid names in the same UPDATE [BEGIN] request will be added to the UPDATE "list" in the usual way.) The RESPONSE ERROR message will be transmitted once, without re-transmissions.

*4

The internal "list" of Information Fields being UPDATE'd by a Controlled Device is cumulative with each UPDATE command.

If a requested Information Field is a time code field, then the first (immediate) UPDATE response will always be in the full Standard Time Code (5-byte) format, while subsequent responses will use the Standard Short Time Code (2-byte) format whenever appropriate. (See Section 3, "Standard Specifications - Standard Short Time Code".)

Use UPDATE [END] to delete items from the "list".

An MMC RESET will completely clear the "list", and halt all UPDATE responses.

43

UPDATE [BEGIN]

<count=variable> Byte count (not including command and count).
 00 "BEGIN" sub-command.
 <name> Information Field name(s) *5

.

.

NOTES:

- *1. If a newly requested Information Field name is already contained in the internal UPDATE "list", then an immediate transmission of the contents of that field will occur as expected. The internal "list" will not change.
- *2. If an Information Field value has changed more than once since the last UPDATE transmission, then only the most recent value will be transmitted.
- *3. If an Information Field value has changed more than once since the last UPDATE transmission, but has reverted to the same value last transmitted, then no new UPDATE response will be required.
- *4. Refer also to the RESPONSE ERROR Information Field description.
- *5. Time code Information Field names may be specified either by their STANDARD TIME CODE names (01 thru 1F), or by their corresponding STANDARD SHORT TIME CODE names (21 thru 3F). Resulting UPDATE actions will be identical in either case.

FORMAT 2 - UPDATE [END]

The Controlled Device must delete the specified Information Field(s) from its UPDATE "list", and discontinue automatic re-transmissions of their contents.

No errors will be generated if the specified name(s) cannot be found in the current UPDATE list.

43	UPDATE [END]
<count=variable>	Byte count (not including command and count).
01	"END" sub-command
<name>	Information Field name(s).

7F anywhere in this list will discontinue all UPDATE's.

NOTES:

1. An MMC RESET command will always empty the UPDATE "list" and cause all UPDATE activity to cease.
2. An UPDATE command which specifies any sub-command other than [BEGIN] or [END], will cause an "Unrecognized sub-command" COMMAND ERROR.

44 LOCATE (MCP command)

FORMAT 1 - LOCATE [I/F]

Causes the Controlled Device to move to the time code position contained in the selected Information Field, in accordance with the SELECTED TIME CODE. *1*2

If the "blank" bit in the target Information Field is set (i.e. a time code value has never been loaded into it), then a "Blank time code" COMMAND ERROR will be generated, and the "MCP Success level" in the MOTION CONTROL TALLY Information Field will be set to "Failure".

With the exception of the DEFERRED PLAY (MCS) and DEFERRED VARIABLE PLAY (MCS) commands, LOCATE [I/F] will be cancelled by the receipt of any other MCS or MCP command.

44	LOCATE [I/F]
<count=02+ext>	Byte count (count=02 where extensions are not used).
00	"I/F" sub-command
<name>	Valid Information Field names are: 00 = reserved for extensions 08 = GP0 / LOCATE POINT 09 = GP1 0A = GP2 0B = GP3 0C = GP4 0D = GP5 0E = GP6 0F = GP7

FORMAT 2 - LOCATE [TARGET]

Causes the Controlled Device to move to the time code position specified in the command data, in accordance with the SELECTED TIME CODE.

With the exception of the DEFERRED PLAY (MCS) and DEFERRED VARIABLE PLAY (MCS) commands, LOCATE [TARGET] will be cancelled by the receipt of any other MCS or MCP command.

44 <count=06> 01 hr mn sc fr ff	LOCATE [TARGET] Byte count. "TARGET" sub-command Standard Time Specification with subframes (type {ff})
--	--

NOTES:

- *1. LOCATE [I/F] requires that at least one of the general purpose registers GP0 thru GP7 be supported.
- *2. Once a LOCATE [I/F] has been initiated, any subsequent changes to the specified Information Field will have no effect on the Locating process. In other words, the locate point time is read from the general purpose register when the LOCATE command is received, and moved to some other unspecified internal locate point register.
3. Devices which do not have the capability to locate with subframe accuracy should ignore any subframes data in the locate point field.
4. At the conclusion of any locating action, if the Controlled Device supports the PAUSE command, then PAUSE mode should be entered. Otherwise, the LOCATE should be concluded with a normal STOP command, with monitoring possibly specified by the STOP MODE Field.
5. Refer also to the descriptions of the DEFERRED PLAY and DEFERRED VARIABLE PLAY commands.
6. A LOCATE command which specifies any sub-command other than [I/F] or [TARGET] will cause an "Unrecognized sub-command" COMMAND ERROR.

45 VARIABLE PLAY (MCS command)

Enter continuously variable playback mode with the direction and speed specified. If the requested speed value exceeds the capabilities of the Controlled Device, then it should play at its "nearest available speed".

VARIABLE PLAY will be cancelled by the receipt of another MCS or MCP command.

45 <count=03> sh sm sl	VARIABLE PLAY Byte count. Standard Speed Specification
------------------------------	--

NOTE:

Recording [rehearsing] tracks do not automatically exit from record [rehearse] upon receipt of the VARIABLE PLAY command. If that action is desired, then transmit <RECORD EXIT> <VARIABLE PLAY>.

46 SEARCH (MCS command)

Causes the Controlled Device to move with the specified direction and velocity.

Output monitoring must be enabled, but need only be of sufficient quality that recorded material is recognizable.

If the requested speed value exceeds the capabilities of the device, then the device should move at its "nearest available speed". In the extreme case, a device which implements only a fixed speed search in each direction can still be said to support the SEARCH command.

SEARCH will be cancelled by the receipt of another MCS or MCP command.
Recording [rehearsing] tracks exit from record [rehearse].

46	SEARCH
<count=03>	Byte count.
sh sm sl	Standard Speed Specification

NOTE:

A device which outputs both picture and audio is only required to monitor picture during a SEARCH. Concurrent audio monitoring remains at the discretion of the equipment manufacturer.

47 SHUTTLE (MCS command)

Causes the Controlled Device to travel at specified direction and velocity without necessarily reproducing audio or picture.

If the requested speed value exceeds the capabilities of the device, then the device should move at its "nearest available speed".

SHUTTLE will be cancelled by the receipt of another MCS or MCP command.

Recording [rehearsing] tracks exit from record [rehearse].

47	SHUTTLE
<count=03>	Byte count.
sh sm sl	Standard Speed Specification

48 STEP (MCS command)

Causes the Controlled Device to move a specified distance forward or backward, with respect to its current position. Successive commands are cumulative until next MCS or MCP command other than STEP.

Visual devices must provide at least visual monitoring during the STEP movement (audio is optional), and must maintain picture after completion of the STEP (similar to a PAUSE mode).

Audio devices must provide audio monitoring during the STEP, and, if the device has digital "looping" capability, should continue to loop after the STEP has been completed.

Sequencers should enable MIDI outputs during the STEP, and should refrain from turning Notes off at completion of the STEP.

In all cases, monitoring should return to "normal" after cancellation of the STEP mode by another MCS or MCP command.

The distance to be moved is measured in units which are defined in the STEP LENGTH Information Field. The default unit is one video field (1/2 frame).

48	STEP
<count=01>	Byte count.
<steps>	Number of STEP LENGTH's: 0 g ssssss
	g = sign (1 = reverse)
	ssssss = quantity

49 ASSIGN SYSTEM MASTER

Most "chase" synchronization systems require definition of a "master" device. Other devices may then follow (chase) and synchronize to the time code from this device.

The assignment of a system master may cause appropriate distribution of that "master" device's time code within the system. Devices receiving such code will normally load it directly or indirectly into their

SELECTED MASTER CODE fields. No particular method of time code distribution or operation is specified by this command.

The assignment of a master time code stream also provides a reference time within which all system-wide timed events may be consistently located.

Reaction to ASSIGN SYSTEM MASTER is governed by the following truth table:

Own <device_ID> = command data?	Device already system master?	ACTION
N	N	none
N	Y	Release system master status
Y	N	Set up as the new system master
Y	Y	Continue as system master

The ASSIGN SYSTEM MASTER message must be transmitted via the "All-Call" device ID (7F). A group device_ID may not be assigned as system master.

49 ASSIGN SYSTEM MASTER
<count=01> Byte count.
<device_ID> Ident of assigned device.
 7F = dis-assign any previously assigned master,
 leaving no master assigned.

4A GENERATOR COMMAND

Controls the running state of the time code generator.
See also the GENERATOR SET UP Information Field.

4A GENERATOR COMMAND
<count=01> Byte count.
nn Action:
 00 = Stop
 01 = Run, frame locked if and as required by the
 GENERATOR SET UP Information Field
 02 = Copy/Jam: While running, transfer Source Time Code
 into the GENERATOR TIME CODE register, as
 specified by the GENERATOR SET UP Information
 Field.

4B MIDI TIME CODE COMMAND

Controls the production of MIDI time code.
See also the MIDI TIME CODE SET UP Information Field.

4B MIDI TIME CODE COMMAND
<count=01> Byte count.
nn Action:
 00 = Off
 02 = Follow the time code defined by the MIDI TIME CODE
 SET UP Information Field.

4C MOVE

Transfer the contents of the Source Information Field to the Destination Information Field.

Valid destination Information Fields are the Read/Writeable fields in the 5-data-byte group (names 01 thru 1F).

Valid source Information Fields are all of the 5-data-byte group (names 01 thru 1F).

Subframes should be assumed to be zero in all sources not usually containing subframes (type {st}).

Subframe data will be lost if the source contains subframes (type {ff}) while the destination does not (type {st}).

4C	MOVE
<count=02+ext>	Byte count (not including command and count).
<destination>	Valid destination Information Field name.
<source>	Valid source Information Field name.

NOTES:

1. All embedded time code status flags (see Section 3 "Standard Specifications") will be transferred from the Source field to the Destination Field, with the exceptions that:
 - (a) When MOVE'ing from an {st} type field to an {ff} field, set bit $i = 0$ as well as subframes = 00;
 - (b) When MOVE'ing from an {ff} field to an {st} field, set bits:
 $i = 1, e = 0, v = 0, d = 0/1$ as determined, and $n = 1$.
2. The MOVE command may be used to instantaneously capture the value of a moving time code stream. For example, to MOVE the current SELECTED TIME CODE to the LOCATE point register GP0; or to trap the current ACTUAL OFFSET by MOVE'ing it to the REQUESTED OFFSET.

4D ADD

Add the contents of two source Information Fields, and place the result in a destination Information Field:
[Destination] = [Source #1] + [Source #2]

The result will be a valid time code number between 00:00:00:00.00 and +/-23:59:59:nn.99, where nn depends on the frame rate used for the calculation. (Whether or not a negative result is permitted depends on the characteristics of the destination Information Field.)

The result will always be expressed as a non-drop-frame quantity, regardless of the drop/non-drop status of either of the sources.

Valid destination Information Fields are the Read/Writeable fields in the 5-data-byte group (01 thru 1F).

Valid source Information Fields are all of the 5-data-byte group (names 01 thru 1F).

It is permissible that the destination Field be the same as one of the source Fields. Care must be exercised so that such source data is not destroyed before the calculation is complete.

Subframes should be assumed to be zero in all sources not usually containing subframes (type {st}).

Subframe data will be lost if either source contains subframes (type {ff}) while the destination does not (type {st}).

4D	ADD
<count=03+ext>	Byte count (not including command and count).
<destination>	Valid destination Information Field name.
<source #1>	Valid source Information Field name.
<source #2>	Valid source Information Field name.

NOTES:

1. The frame rate and drop-frame status of each of the source fields is established entirely by the time code status bits embedded within those fields. The TIME STANDARD Information Field has no bearing on this calculation.

2. Embedded time code status flags of an *{ff}* type Destination field (see Section 3 "Standard Specifications") will be set up as follows:

<i>tt</i> (time type)	Same as <u>Source #1</u> , with the exception that if Source #1 specifies drop-frame, then the Destination will be converted to non-drop-frame.
-----------------------	---

<i>c</i> (color frame)	0
<i>k</i> (blank)	0
<i>g</i> (sign)	Set by result of calculation. If the Destination field does not allow signed negative data, then any negative result must first be added to "24 hours" to produce a positive value.

<i>i</i> (final byte id)	0
--------------------------	---

3. Embedded time code status flags of an *{st}* type Destination field (see Section 3 "Standard Specifications") will be set up as follows:

<i>tt, c, k, g</i>	Same as <i>{ff}</i> field
<i>i</i> (final byte id)	1
<i>e</i> (estimated code)	0
<i>v</i> (invalid code)	0
<i>d</i> (video field 1)	0/1 as determined
<i>n</i> (no time code)	1

4. It is expected that many devices will be capable of performing mixed 30 frame drop and non-drop calculations. Very few, however, will produce correct results with other frame rate mis-matches.

5. Calculations involving drop frame code which "cross" the 24 hour boundary may produce unpredictable results.

4E SUBTRACT

Subtract the contents of one source Information Field from that of the other, and place the result in a destination Information Field:

$$[\text{Destination}] = [\text{Source } \#1] - [\text{Source } \#2]$$

All the conditions for the ADD command apply also to the SUBTRACT command.

4E

<count=03+ext>
<destination>
<source #1>
<source #2>

SUBTRACT

Byte count (not including command and count).
Valid destination Information Field name.
Valid source Information Field name.
Valid source Information Field name.

4F DROP FRAME ADJUST

Convert the contents of the named Information Field into drop-frame format (see "Drop Frame Notes" in the "Standard Specifications" section).

Take no action if the contents are not currently expressed in 30 frame, non-drop-frame format.

May be used after ADD or SUBTRACT to produce a drop frame result.

Valid Information Fields are the Read/Writeable fields in the 5-data-byte group (names 01 thru 1F).

4F <count=01+ext> <name>	DROP FRAME ADJUST Byte count (not including command and count). Valid Information Field name
---	---

NOTE:

Frame rate and drop frame status of named Information Field is established entirely by the time code status bits contained within that field. The TIME STANDARD Information Field has no bearing on this calculation.

50 PROCEDURE

A Procedure is a string of commands defined by the Controller and stored within the Controlled Device. It may subsequently be executed by transmission of a single PROCEDURE [EXECUTE] command.

FORMAT 1 - PROCEDURE [ASSEMBLE]:

Assembles a string of commands for execution at a later time.

Procedures are retained until the receipt of a PROCEDURE [DELETE] or MMC RESET command. Re-definition of an already defined procedure implies that the previous definition first be deleted.

The commands contained within a procedure must be pre-checked for "MAJOR" and "IMPLEMENTATION" errors when the procedure is assembled. If these embedded commands contain such errors, then: *1

- (i) the procedure will be discarded;
- (ii) the error will be recorded in the COMMAND ERROR Information Field;
- and (iii) the PROCEDURE [ASSEMBLE] error flag will be set (also in the COMMAND ERROR Information Field).

Any attempt to define a procedure which will overflow whatever procedure storage area is available will generate a "PROCEDURE buffer overflow" error in the COMMAND ERROR Information Field.

50 <count=variable> 00 <procedure> <command #1..> <command #2..> <command #3..>	PROCEDURE [ASSEMBLE] Byte count. "ASSEMBLE" sub-command. Procedure Name in the range 00 thru 7E. 7F is reserved. Any MMC commands except: (i) another PROCEDURE [ASSEMBLE]; *2 or (ii) a PROCEDURE [EXECUTE] with the same procedure name as is being defined. *3
--	---

NOTES:

- *1. "MAJOR" and "IMPLEMENTATION" errors are described in the COMMAND ERROR Information Field definition. Refer also to NOTE 2 of that definition.
- *2. A "Nested PROCEDURE [ASSEMBLE]" error would be generated.
- *3. A "Recursive PROCEDURE [EXECUTE]" error would be generated.

FORMAT 2 - PROCEDURE [DELETE]:

Delete a previously defined sequence of commands. With the exception of the "delete all PROCEDURE's" version of this command, any attempt to delete an undefined procedure will cause an "Undefined PROCEDURE" error in the COMMAND ERROR Information Field.

50	PROCEDURE [DELETE]
<count=02>	Byte count.
01	"DELETE" sub-command.
<procedure>	Procedure Name in the range 00 thru 7E. 7F means <u>delete all PROCEDURE's</u> .

FORMAT 3 - PROCEDURE [SET]:

Establishes the name of the procedure which will appear in the next READ of the PROCEDURE RESPONSE Information Field. With the exception of the "set all PROCEDURE's" version of this command, specification of an undefined procedure will cause an "Undefined PROCEDURE" error in the COMMAND ERROR Information Field.

50	PROCEDURE [SET]
<count=02>	Byte count.
02	"SET" sub-command.
<procedure>	Procedure Name in the range 00 thru 7E. 7F means <u>set all PROCEDURE's</u> .

FORMAT 4 - PROCEDURE [EXECUTE]:

Immediately execute the named procedure. Any attempt to execute an undefined procedure will cause an "Undefined PROCEDURE" error in the COMMAND ERROR Information Field.

50	PROCEDURE [EXECUTE]
<count=02>	Byte count.
03	"EXECUTE" sub-command.
<procedure>	Procedure Name in the range 00 thru 7E. 7F is reserved.

NOTES:

1. An MMC RESET command will always delete all Procedures.
2. A PROCEDURE command which specifies any sub-command other than [ASSEMBLE], [DELETE], [SET] or [EXECUTE], will cause an "Unrecognized sub-command" COMMAND ERROR.
3. Examples of PROCEDURE [ASSEMBLE] and PROCEDURE [EXECUTE] commands may be found in Note 8 of the EVENT [DEFINE] command description.

51 EVENT

FORMAT 1 - EVENT [DEFINE]

Allows any single MIDI Machine Control command to be executed by the Controlled Device at a specified trigger time, relative to a specified time code stream.

Re-definition of an already defined Event implies that the previous definition first be deleted.

Any attempt to define an Event which will overflow whatever Event storage area is available will generate an "EVENT buffer overflow" error in the COMMAND ERROR Information Field.

Similarly, if the requested "trigger source" Information Field is unavailable for any reason, an "EVENT trigger source unavailable or unsupported" error will be generated, and the Event will be discarded.

The command contained within the Event must be pre-checked for "MAJOR" and "IMPLEMENTATION" errors when the Event is defined. If this embedded command contains such errors, then: *9

- (i) the Event will be discarded;
- (ii) the error will be recorded in the COMMAND ERROR Information Field;
- and (iii) the EVENT [DEFINE] error flag will be set
(also in the COMMAND ERROR Information Field).

```
51           EVENT [DEFINE]
<count=variable> Byte count.
00           "DEFINE" sub-command.
<event>       Event Name in the range 00 thru 7E.
                7F is reserved.
<flags>        Event control flags: 0 k 0 a 00 dd
                dd = Direction modes:
                00 = Trigger only while moving forwards.
                01 = Trigger only while moving in reverse.
                10 = Trigger while moving in either direction.
                a = "All speeds" flag:
                0 = Trigger only when trigger time is equaled while
                    moving at fixed or variable play-speed.
                1 = Trigger immediately upon recognizing that the
                    trigger time has been equaled or passed,
                    while moving at any speed.
                k = "Non-delete" flag:
                0 = Delete Event definition immediately upon being
triggered (ESbus mode).
                1 = Event definition remains in event queue after
                    being triggered. *8
<trigger source> Information Field name of time code stream relative to which the event
is to be triggered: *4
00 = reserved for extensions
01 = SELECTED TIME CODE *5
02 = SELECTED MASTER CODE
06 = GENERATOR TIME CODE
07 = MIDI TIME CODE INPUT
<name>        Name of Information Field containing the trigger time: *3
00 = reserved for extensions
08 = GP0 / LOCATE POINT
09 = GP1
0A = GP2
0B = GP3
0C = GP4
```

<code><command..></code>	<code>0D = GP5</code> <code>0E = GP6</code> <code>0F = GP7</code> Any single command plus data as required, with the exception of: *6 (i) another EVENT [DEFINE]; *10 or (ii) a PROCEDURE [ASSEMBLE]. *11
--------------------------------	--

NOTES:

1. The EVENT command requires that at least one of the general purpose registers GP0 thru GP7 be supported.
2. Any subsequent changes to the specified trigger time register will have no effect on the Event. In other words, the trigger time is read from the general purpose register when the Event is defined, and moved an unspecified internal Event trigger time area. An EVENT RESPONSE will always send back the time from this internal area.
- *3. Whether or not to support subframe accurate Event triggering remains at the discretion of the device manufacturer.
- *4. Typical trigger sources will be SELECTED TIME CODE or SELECTED MASTER CODE. Limitations may occur in some devices which only support a single trigger source (probably SELECTED TIME CODE). Other devices may support multiple trigger sources, but only implement subframe triggering for one or more of them.
- *5. If no time code is present, and SELECTED TIME CODE is always updated by tachometer pulses, then it may not provide an ideal trigger source, as its numeric sequence can be discontinuous. This problem may be circumvented by defining Events which may be triggered at "All speeds" (refer to the EVENT command decription).
- *6. In order to execute multiple commands at a time, the command PROCEDURE [EXECUTE] should be used.
7. It is important that MIDI Machine Control commands which may require advance triggering should be detected within the Controlled Device, and their trigger times advanced accordingly (for example, RECORD STROBE must allow for record ramp up delays). This action should be transparent to the Controller.
- *8. Example of an Event in the "Non-delete" mode:
Consider a simple "looping" action where a tape machine is to begin playing from location "A" and continue up to point "B", at which time it must locate back to "A" and start again:

```

F0 7F <device_ID> <mcc>
  <WRITE> <count=0C>
    <GP0> <5-byte loop start time "A">
    <GP1> <5-byte loop end time "B">
  <PROCEDURE> <count=06> <[ASSEMBLE]> <procedure_name>
    <LOCATE> <count=01> <GP0>
    <DEFERRED PLAY>
  <EVENT> <count=09> <[DEFINE]> <event_name>
    <flags=40> <SELECTED TIME CODE> <GP1>
    <PROCEDURE> <count=02> <[EXECUTE]> <procedure_name>
    <PROCEDURE> <count=02> <[EXECUTE]> <procedure_name>

```

F7

- *9. "MAJOR" and "IMPLEMENTATION" errors are described in the COMMAND ERROR Information Field definition. Refer also to NOTE 2 of that definition.
- *10. A "Nested EVENT [DEFINE]" error would be generated.
- *11. A "PROCEDURE [ASSEMBLE] within EVENT [DEFINE]" error would be generated.

FORMAT 2 - EVENT [DELETE]

Delete a previously defined Event.

With the exception of the "delete all EVENT's" version of this command, any attempt to delete an undefined event will cause an "Undefined EVENT" error in the COMMAND ERROR Information Field.

51	EVENT [DELETE]
<count=02>	Byte count.
01	"DELETE" sub-command.
<event>	Event Name in the range 00 thru 7E. 7F means <u>delete all EVENT's</u> .

FORMAT 3 - EVENT [SET]

Establishes the name of the Event which will appear in the next READ of the EVENT RESPONSE Information Field.

With the exception of the "set all EVENT's" version of this command, specification of an undefined Event will cause an "Undefined EVENT" error in the COMMAND ERROR Information Field.

51	EVENT [SET]
<count=02>	Byte count.
02	"SET" sub-command.
<event>	Event Name in the range 00 thru 7E. 7F means <u>set all EVENT's</u> .

FORMAT 4 - EVENT [TEST]

Immediately execute the command contained in the named Event, as if a trigger had occurred.

Do not delete the Event.

Any attempt to test an undefined Event will cause an "Undefined EVENT" error in the COMMAND ERROR Information Field.

51	EVENT [TEST]
<count=02>	Byte count.
03	"TEST" sub-command.
<event>	Event Name in the range 00 thru 7E 7F is reserved.

NOTES:

1. An MMC RESET command will always delete all Events.
2. An EVENT command which specifies any sub-command other than [DEFINE], [DELETE], [SET] or [TEST], will cause an "Unrecognized sub-command" COMMAND ERROR.

52 GROUP

FORMAT 1 - GROUP [ASSIGN]

The Controlled Device is to become assigned to the indicated Group if the device's <device_ID> appears in the received list of device_ID's.

Once assigned, the Controlled Device is to honour all commands received via the Group device_ID in addition to those received through its own device_ID.

A Group assignment is retained until receipt of an MMC RESET or an appropriate GROUP [DIS-ASSIGN] command.

Group assignment is cumulative with each GROUP [ASSIGN] message. For example, to add a new device to an already existing group, a Controller may simply transmit a GROUP [ASSIGN] listing only the new device.

Any attempt to assign the device to more groups than it can accomodate will produce a "Group buffer overflow" error in the COMMAND ERROR Information Field. *3

52	GROUP [ASSIGN]
<count=variable>	Byte count (not including command and count).
00	"ASSIGN" sub-command.
<group>	Group number - any <u>unused</u> device_ID may be assigned as a group number, with the exception of 7F.
<device_ID>	List of devices which are to begin responding to the Group number ..
<device_ID>	
.	
.	
.	

FORMAT 2 - GROUP [DIS-ASSIGN]

The Controlled Device should remove itself from the indicated Group if the device's <device_ID> appears in the received list of device_ID's.

52	GROUP [DIS-ASSIGN]
<count=variable>	Byte count.
01	"DIS-ASSIGN" sub-command.
<group>	Group number 7F = dis-assign from <u>all</u> groups.
<device_ID>	List of devices which are to be dis-assigned from the Group number ..
<device_ID>	7F anywhere in this list will dis-assign <u>all</u> devices. *4
.	
.	
.	

NOTES:

1. An MMC RESET command will always delete all Group assignments.
2. GROUP [ASSIGN] and [DIS-ASSIGN] messages will normally be transmitted via the "all-call" device_ID (<device_ID> = 7F).
- *3. It is recommended that a Controlled Device should accommodate being assigned to at least 16 groups at any one time.
- *4. When dis-assigning an entire group, the "list of devices" will normally contain only a single entry (7F). For example, to dis-assign all devices from all groups, the Controller transmits:
F0 7F 7F <mcc> <GROUP> <count=03> 01 7F 7F F7
5. A GROUP command which specifies any sub-command other than [ASSIGN] or [DIS-ASSIGN], will cause an "Unrecognized sub-command" COMMAND ERROR.

53 COMMAND SEGMENT

Allows a command (or a string of commands), which is greater in length than the maximum MMC System Exclusive data field length (48 bytes), to be divided into segments and transmitted piece by piece across multiple System Exclusives.

Commands received by a Controlled Device in this way will be executed exactly as if they had arrived all in the same sysex.

COMMAND SEGMENT must always be the first command in its sysex, and there must be no other commands in the sysex save those which are contained within the body of COMMAND SEGMENT itself.

Segment divisions need not fall on command boundaries. Partial commands, which may occur at the end of a COMMAND SEGMENT sysex, must be detected by the Controlled Device so that command processing may be correctly resumed when the next segment arrives.

A Controlled Device will generate a "Segmentation Error", one of the "MAJOR ERRORS" defined in the COMMAND ERROR Information Field, under any of the following conditions:

- (a) COMMAND SEGMENT not first command in sysex;
- or (b) Byte count not exactly equal to number of bytes remaining in sysex;
- or (c) A "subsequent" segment is received without receiving a "first" segment;
- or (d) Segments are received out of order;

De-segmentation will be cancelled upon the occurrence of a Segmentation Error.

With the exception of WAIT or RESUME messages, if a non-segmented (i.e. normal) sysex is received by a Controlled Device when a "subsequent" segment sysex was expected, it will be processed normally, and de-segmentation will be cancelled.

Refer also to Section 2 "General Structure" - "Segmentation".

53	COMMAND SEGMENT
<count=variable>	Byte count (command string segment length + 1)
si	Segment Identification: 0 f ssssss
	f: 1 = first segment
	0 = subsequent segment
	ssssss = segment number (down count, last=000000)
<.. commands ..>	Command string segment.

54 DEFERRED VARIABLE PLAY (MCS command)

Identical to the VARIABLE PLAY command, with the exception that if the device is currently executing a LOCATE (MCP), then VARIABLE PLAY mode will not be invoked until the LOCATE is completed.

Receipt of any other MCS or MCP command will cancel DEFERRED VARIABLE PLAY.

When received while a LOCATE is in progress, the "MCP Success Level" field of the MOTION CONTROL TALLY Information Field will be set to indicate "Actively locating with Deferred Variable Play pending" for the duration of the LOCATE.

When the LOCATE has concluded:

- (i) An automatic VARIABLE PLAY (MCS) command will be issued, and the device will enter continuously variable playback mode with the direction and speed specified (If the requested speed value exceeds the capabilities of the Controlled Device, then it should play at its "nearest available speed");
- (ii) The MOTION CONTROL TALLY "MCS command" byte will switch to "VARIABLE PLAY";
- (iii) The MOTION CONTROL TALLY "MCP command" byte will become "No MCP's currently active", clearing both the LOCATE and Deferred Variable Play indications.

If the device is not executing or does not support the LOCATE command, then it should immediately enter variable playback mode.

54 DEFERRED VARIABLE PLAY
<count=03> Byte count.
sh sm sl Standard Speed Specification

NOTE:

Recording [rehearsing] tracks do not automatically exit from record [rehearse] upon receipt of the DEFERRED VARIABLE PLAY command. If that action is desired, then transmit <RECORD EXIT> <DEFERRED VARIABLE PLAY>.

55 RECORD STROBE VARIABLE

Switches the Controlled Device into or out of record or rehearse, according to the setting of the RECORD MODE Information Field. RECORD STROBE VARIABLE will be honored under two conditions only:

CONDITION 1: Controlled Device already Playing:

If the Controlled Device is already playing (i.e. the "Most recently activated Motion Control State" in the MOTION CONTROL TALLY Information Field is PLAY or VARIABLE PLAY), then RECORD STROBE VARIABLE will cause record [rehearse] entry on all tracks which are presently in a record ready state, and cause record [rehearse] exit on any currently recording [rehearsing] tracks that are no longer record ready. *2*9*10

CONDITION 2: Controlled Device Stopped:

If, when RECORD STROBE VARIABLE is received, the Controlled Device is completely stopped as a result of an explicit STOP or PAUSE command (i.e. [i] the "Most recently activated Motion Control State" in the MOTION CONTROL TALLY Information Field is STOP or PAUSE; [ii] the "MCS success level" shows "Completely stopped"; and [iii] the "Most recently activated Motion Control Process" byte is set to "No MCP's currently active"), then:

- (i) An automatic VARIABLE PLAY (MCS) command will be issued, and the device will enter continuously variable playback mode with the direction and speed specified. (If the requested speed value exceeds the capabilities of the Controlled Device, then it should play at its "nearest available speed"); *4*5
- (ii) At an appropriate point in the varispeed start up phase of the device, record [rehearse] entry will occur on all tracks which are presently in a record ready state. *2*6

55 <count=03> sh sm s1	RECORD STROBE VARIABLE Byte count. Standard Speed Specification
------------------------------	---

NOTES:

1. The recording [rehearsing] characteristics of RECORD STROBE VARIABLE are identical to those of the RECORD STROBE command. The only difference between the two commands lies in the nature of the play mode command which is invoked automatically if the Controlled Device is completely stopped. RECORD STROBE VARIABLE will therefore be used if variable speed recording [rehearsing] must be achieved from a standing start.
- *2. Tracks are switched in and out of the record ready state using the TRACK RECORD READY Information Field.
3. It is recommended that, for new Controlled Device designs based on MMC, no recording [rehearsing] should take place following a RECORD STROBE VARIABLE command unless at least one track is in a record ready state.
Among existing non-MMC devices, however, it is quite common that, given that record [rehearse] has been enabled (for example by setting the appropriate value in the RECORD MODE Information Field), recording [rehearsing] will be initiated even if no tracks are in a record ready state when the command to record [rehearse] is received. Such operation remains permissible under MMC, provided that the resultant status is correctly indicated by including the "No Tracks Active" bit in the RECORD STATUS Information Field.
- *4. Under CONDITION 2, an automatic VARIABLE PLAY (MCS) command will be issued only under the STOP or PAUSE conditions specified. At no other time does RECORD STROBE VARIABLE have any implications regarding play mode or playing speed.
- *5. Also under CONDITION 2, the automatic VARIABLE PLAY (MCS) command will be issued whether or not any tracks are in a record ready state, and whether or not a RECORD MODE has been specified.
- *6. The existence of a "record-pause" condition will not affect the operation of RECORD STROBE VARIABLE. (Refer to the RECORD PAUSE and PAUSE command descriptions.)
7. Devices which support and have their RECORD MODE set to "VTR:Insert" or "VTR:Assemble", are expected to produce clean and correctly timed transitions into record [rehearse] under both Conditions 1 and 2 outlined above. When starting from STOP or PAUSE mode (Condition 2), it will usually be necessary to wait until the device's start up phase has been completed before recording [rehearsing] is attempted.
8. A Controlled Device will ignore any RECORD STROBE VARIABLE command which is received while it is neither already in play mode nor completely stopped as described.
- *9. Under CONDITION 1, "Controlled Device already Playing", it is not necessary for the PLAY or VARIABLE PLAY command to have been "successful" before RECORD STROBE VARIABLE is accepted. If, however, the desired play motion has not yet been achieved when RECORD STROBE VARIABLE is received, it may be necessary for the device to defer the onset of recording until an appropriate point in its start up phase.
- *10. Note also that, under CONDITION 1, recording is not inhibited by Motion Control Process activity.

7C WAIT

Signals the Controlled Device that the Controller's receive buffer is filling (or that the Controller is otherwise unavailable), and that Machine Control Response transmissions must be discontinued until receipt of a RESUME from the Controller. Any Response transmission which is currently in progress will be allowed to proceed up to its normal End of System Exclusive (F7). Transmission of subsequent Responses may resume after receipt of a RESUME from the Controller.
The Responses WAIT and RESUME, however, are not inhibited by the WAIT Command. Neither is transmission of the WAIT Command itself inhibited by receipt of a WAIT Response.

A Controlled Device must guarantee:

- and (i) to recognize the receipt of a WAIT message within 10 milliseconds after the arrival of the End of System Exclusive (F7) of that WAIT message;
 (ii) to then halt all transmissions at the next available MMC System Exclusive boundary (up to 53 bytes, the maximum MMC sysex length, may therefore have to be transmitted before the halt can take effect).

The WAIT command is always the only command in its Sysex, and is directed to the "all-call" address i.e. F0 7F 7F <mcc> <WAIT> F7.

7C WAIT

NOTES:

1. Correct operation of the WAIT command requires a certain minimum size for the MIDI receive buffer in the Controller. Refer to Appendix E, "Determination of Receive Buffer Size".
2. Additional WAIT commands may be transmitted by a Controller should its receive buffer continue to fill.

7F RESUME

Signifies that the Controller is ready to receive Machine Control Responses from the Controlled Device. The default (power up) state is "ready to receive".

The RESUME command is used primarily to allow the Controlled Device to resume transmissions after a WAIT.

Transmission of the RESUME Command is not inhibited by the receipt of a WAIT Response.

The RESUME command is always the only command in its Sysex, and is directed to the "all-call" address i.e. F0 7F 7F <mcc> <RESUME> F7.

7F RESUME

6 DETAILED RESPONSE & INFORMATION FIELD DESCRIPTIONS

Messages from CONTROLLED DEVICE to CONTROLLER.

00 Reserved for extensions

01 SELECTED TIME CODE [read/write]

Contains the time code normally used to reference the Controlled Device's current position. (It may also be referred to as "Self" code, or "Slave" time code.)

The Information Field SELECTED TIME CODE SOURCE determines the source of this time code. It is selected from Longitudinal Time Code (LTC), Vertical Interval Time Code (VITC), and the "tape counter" found on most tape machines.

01	SELECTED TIME CODE
<i>hr mn sc fr st</i>	Standard Time Specification with status (type {st})

NOTES:

1. More details concerning time code choices may be found in the SELECTED TIME CODE SOURCE Information Field description.
2. The SELECTED TIME CODE status byte will indicate whether or not the current time code has been updated by Tachometer or Control Track pulses, for example, during fast wind modes.
3. If no time code is available at all, then SELECTED TIME CODE will be equivalent to the "tape counter" which is found on most tape machines, usually updated by Tachometer or Control Track pulses. For compatibility in time code systems, this "tape counter" must count in hours, minutes, seconds and frames. The time code mode for this counter will normally be determined either by the TIME STANDARD Information Field, if supported, or by some other form of locally adjustable default. If, however, the Controller WRITE's a value into SELECTED TIME CODE, then the time code mode will be determined by the *tt* (time type) field in the WRITE data. Negatively signed values of SELECTED TIME CODE are not permitted.
4. SELECTED TIME CODE is specified as WRITE-capable in order to adequately support the above "tach only" mode of operation. In this case, setting the SELECTED TIME CODE "counter" to a new value is an accepted operational procedure. However, when time code is available from the tape, WRITE'ing a new value to this Field may produce unexpected results.
5. It is not expected that synchronization will be attempted unless "real" time code is available in SELECTED TIME CODE (i.e. time code status bit *n* = 0).
6. Refer to Appendix B, "Time Code Status Implementation Tables" for exact usage of all embedded time code status bits as they apply to SELECTED TIME CODE.
Refer to Section 3, "Standard Specifications", for definition of these bits.

02 SELECTED MASTER CODE [read only]

Contains the time value of the time code relative to which all synchronization operations are to take place (see the CHASE command). How this time code arrives at the Controlled Device is not specified.

02	SELECTED MASTER CODE
<i>hr mn sc fr st</i>	Standard Time Specification with status (type {st})

NOTES:

1. Future versions of the MIDI Machine Control specification may provide a method of selecting this MASTER CODE from a number of specific time code sources.
2. Refer to Appendix B, "Time Code Status Implementation Tables" for exact usage of all embedded time code status bits as they apply to SELECTED MASTER CODE.
Refer to Section 3, "Standard Specifications", for definition of these bits.

03 REQUESTED OFFSET [read/write]

Contains the desired time offset between the SELECTED TIME CODE and the SELECTED MASTER CODE for use with the CHASE command, and is defined as follows:

REQUESTED OFFSET = SELECTED TIME CODE - SELECTED MASTER CODE

[Example: If the Controlled Device is to lead the Master Device by one minute, then
REQUESTED OFFSET = 00:01:00:00.00]

This offset represents the desired difference in frames between the master and slave positions, and is always expressed as a non-drop-frame number.

REQUESTED OFFSET may be expressed in any positive or negative range. MMC devices will interpret an offset of +23:00:00:00.00, for example, as being equivalent to one of -01:00:00:00.00.

03	REQUESTED OFFSET
hr mn sc fr ff	Standard Time Specification with subframes (type {ff})

NOTE:

Refer to Appendix B, "Time Code Status Implementation Tables" for exact usage of all embedded time code status bits as they apply to REQUESTED OFFSET.

Refer to Section 3, "Standard Specifications", for definition of these bits.

04 ACTUAL OFFSET [read only]

For synchronization purposes, this field contains the actual time difference between the current values of SELECTED MASTER CODE and SELECTED TIME CODE, where:

ACTUAL OFFSET = SELECTED TIME CODE - SELECTED MASTER CODE

[Example: If Controlled Device leads Master Device by one minute, then the ACTUAL OFFSET is 00:01:00:00.00]

This offset represents the difference in frames between the slave and master positions and is always expressed as a non-drop-frame number.

ACTUAL OFFSET must be expressed in the range +/-12:00:00:00.00, based on the "time code equivalence" of numbers such as -01:00:00:00.00 and +23:00:00:00.00.

04	ACTUAL OFFSET
hr mn sc fr ff	Standard Time Specification with subframes (type {ff})

NOTE:

Refer to Appendix B, "Time Code Status Implementation Tables" for exact usage of all embedded time code status bits as they apply to ACTUAL OFFSET.

Refer to Section 3, "Standard Specifications", for definition of these bits.

05 LOCK DEVIATION [read only]

For synchronization purposes, this field contains the time difference between the position of the Controlled Device's SELECTED TIME CODE and the SELECTED MASTER CODE after adjustment by the REQUESTED OFFSET:

LOCK DEVIATION = ACTUAL OFFSET - REQUESTED OFFSET

or

LOCK DEVIATION = SELECTED TIME CODE - SELECTED MASTER CODE - REQ'D OFFSET

LOCK DEVIATION is always a non-drop-frame number and must be expressed in the range +/-12:00:00:00.00.

05 LOCK DEVIATION
hr mn sc fr ff Standard Time Specification with subframes (type {ff})

NOTE:

Refer to Appendix B, "Time Code Status Implementation Tables" for exact usage of all embedded time code status bits as they apply to LOCK DEVIATION.

Refer to Section 3, "Standard Specifications", for definition of these bits.

06 GENERATOR TIME CODE [read/write]

Contains the current time code value being generated by the time code generator.

06 GENERATOR TIME CODE
hr mn sc fr st Standard Time Specification with status (type {st})

NOTE:

Refer to Appendix B, "Time Code Status Implementation Tables" for exact usage of all embedded time code status bits as they apply to GENERATOR TIME CODE.

Refer to Section 3, "Standard Specifications", for definition of these bits.

07 MIDI TIME CODE INPUT [read only]

Contains the most recent incoming MIDI Time Code value.

07 MIDI TIME CODE INPUT
hr mn sc fr st Standard Time Specification with status (type {st})

NOTE:

Refer to Appendix B, "Time Code Status Implementation Tables" for exact usage of all embedded time code status bits as they apply to MIDI TIME CODE INPUT.

Refer to Section 3, "Standard Specifications", for definition of these bits.

08 GP0 / LOCATE POINT [read/write]

General Purpose time code and calculation register 0.

08 GP0 / LOCATE POINT
hr mn sc fr ff Standard Time Specification with subframes (type {ff})

NOTES:

1. The LOCATE [I/F] command specifies that a General Purpose register must contain its target location time. Similarly, the EVENT command takes its trigger time from a General Purpose register. Therefore, at least one General Purpose register must be supported if either the LOCATE or the EVENT commands are to be used.
2. General Purpose registers may be employed to capture moving time code "on the fly" (for example by MOVE'ing the SELECTED TIME CODE to GPn). This can also remove the need for the Controller to always read back actual time code values, thus facilitating operation in the "open loop" mode.
3. Signed time code is permitted in a General Purpose register.
4. Refer to Appendix B, "Time Code Status Implementation Tables" for exact usage of all embedded time code status bits as they apply to the General Purpose registers.
Refer to Section 3, "Standard Specifications", for definition of these bits.

09	GP1 [read/write]
0A	GP2 [read/write]
0B	GP3 [read/write]
0C	GP4 [read/write]
0D	GP5 [read/write]
0E	GP6 [read/write]
0F	GP7 [read/write]

General Purpose time code and calculation registers 1 thru 7.

(See notes for General Purpose register 0)

<name> GP1 thru GP7
hr mn sc fr ff Standard Time Specification with subframes (type {ff})

21	Short SELECTED TIME CODE [read only]
22	Short SELECTED MASTER CODE [read only]
23	Short REQUESTED OFFSET [read only]
24	Short ACTUAL OFFSET [read only]
25	Short LOCK DEVIATION [read only]
26	Short GENERATOR TIME CODE [read only]
27	Short MIDI TIME CODE INPUT [read only]
28	Short GP0 / LOCATE POINT [read only]
29	Short GP1 [read only]
2A	Short GP2 [read only]
2B	Short GP3 [read only]
2C	Short GP4 [read only]
2D	Short GP5 [read only]
2F	Short GP6 [read only]
2F	Short GP7 [read only]

Refer to Section 3, "Standard Specifications", for definition of "Standard Short Time Code". In each case, refer also to the corresponding 5-byte time code Information Field for description of data content.

<code><name></code>	Short time code Information Field name
<code>fr {st/ff}</code>	Standard Short Time Code Specification

40 SIGNATURE [read only]

Dual bitmap array of (a) all Command functions and (b) all Responses/Information Fields supported by the Controlled Device.

In all cases, bits are set to 1 if the corresponding functions are supported, or partially supported.

The SIGNATURE Information Field for a Controlled Device must be published by its manufacturer, using the format shown in Note *5.

<code>40</code>	SIGNATURE
<code><count=variable></code>	Byte count of all subsequent data. *1
<code>vi</code>	MMC Version implemented by the device; integer part, converted to binary. For the current version <code>vi=01</code> .
<code>vf</code>	MMC Version implemented by the device; fractional part, 00 thru 99, converted to binary (00-63h). For current version, <code>vf=00</code> .
<code>va</code>	reserved, must be set to 00 (MMC version extension)
<code>vb</code>	reserved, must be set to 00 (MMC version extension)
<code><count_1></code>	Byte count for <u>Command Bitmap Array</u>
<code>c0</code>	Command bitmap 0: Commands 00 thru 06: 0 <code>gfedcba</code> <code>a</code> = Command 00 <code>b</code> = Command 01 <code>c</code> = Command 02 <code>d</code> = Command 03 <code>e</code> = Command 04 <code>f</code> = Command 05 <code>g</code> = Command 06
<code>c1</code>	Command bitmap 01: Commands 07 thru 0D
<code>c2</code>	Command bitmap 02: Commands 0E thru 14
<code>c3</code>	Command bitmap 03: Commands 15 thru 1B
<code>c4</code>	Command bitmap 18: Commands 1C thru 1F: 0000 <code>dcba</code> <code>a</code> = command 1C <code>b</code> = command 1D <code>c</code> = Command 1E <code>d</code> = Command 1F
<code>c5</code>	Command bitmap 05: Commands 20 thru 26
<code>c6</code>	Command bitmap 06: Commands 27 thru 2D
<code>c7</code>	Command bitmap 07: Commands 2E thru 34
<code>c8</code>	Command bitmap 08: Commands 35 thru 3B
<code>c9</code>	Command bitmap 09: Commands 3C thru 3F

<i>c10</i>	Command bitmap 10: Commands 40 thru 46
<i>c11</i>	Command bitmap 11: Commands 47 thru 4D
<i>c12</i>	Command bitmap 12: Commands 4E thru 54
<i>c13</i>	Command bitmap 13: Commands 55 thru 5B
<i>c14</i>	Command bitmap 14: Commands 5C thru 5F
<i>c15</i>	Command bitmap 15: Commands 60 thru 66
<i>c16</i>	Command bitmap 16: Commands 67 thru 6D
<i>c17</i>	Command bitmap 17: Commands 6E thru 74
<i>c18</i>	Command bitmap 17: Commands 75 thru 7B
<i>c19</i>	Command bitmap 17: Commands 7C thru 7F
<i>c20 thru c39</i>	Command bitmaps 20 thru 39: Extended commands 00 01 thru 00 7F
<i>c40 thru c59</i>	Command bitmaps 40 thru 59: Extended commands 00 00 01 thru 00 00 7F
<i><count_2></i>	Byte count for <u>Response/Information Field Array</u>
<i>r0 thru r19</i>	Response/Information Field bitmaps 00 thru 19: Responses and Information Fields 00 thru 7F
<i>r20 thru r39</i>	Response/Information Field bitmaps 20 thru 39: Extended Responses and Information Fields 00 01 thru 00 7F
<i>r40 thru r59</i>	Response/Information Field bitmaps 40 thru 59: Extended Responses and Information Fields 00 00 01 thru 00 00 7F

NOTES:

- *1. The maximum value for *<count>*, when both extension sets are fully supported, is 7E.
- 2. Transmit as many bytes in each array as are required.
- 3. Commands and Responses/Information Fields not included in the transmission are assumed to be unsupported.
- 4. In addition to this SIGNATURE, all devices should support the MIDI Inquiry message.
- *5. When published, the SIGNATURE will appear in the following format:

```

vi vf va vb
<count_1>
c0 c1 c2 c3 c4 c5 c6 c7 c8 c9
c10 c11 c12 c13 c14 c15 c16 c17 c18 c19
c20 c21 c22 etc.
<count_2>
r0 r1 r2 r3 r4 r5 r6 r7 r8 r9
r10 r11 r12 r13 r14 r15 r16 r17 r18 r19
r20 r21 r22 etc.

```

(Refer also to Appendix C "Signature Table".)

- 6. All Controlled Devices will show Command 00 as being supported, and will have the capability to correctly parse extended commands, even if none are supported.
- Response 00, on the other hand, will always be shown as unsupported in the current version. In future versions, the Response 00 bit will be the logical "OR" of all extended Response bits.

41 UPDATE RATE [read/write]

Establishes a minimum time between repetitive UPDATE transmission cycles.
Refer to the UPDATE command description.

41	UPDATE RATE
<count=01>	Byte count.
<interval>	Minimum time interval between repetitive UPDATE transmissions expressed as a 7 bit frame count. Default interval is one frame (<interval=01>).

42 RESPONSE ERROR [no access]

Every READ or UPDATE request for a particular Information Field must generate a response from the Controlled Device. If, however, a requested Information Field is:

- (a) unsupported by the Controlled Device,
- or (b) undefined within MMC,
- or (c) defined by MMC as having "no access",

then a RESPONSE ERROR message will be substituted for the expected Information Field response.

A READ command with a list of "n" different Information Fields will be treated as "n" different requests, all of which must be responded to. The same is true for the UPDATE command.

If desired, several unsupported Information Field names may be gathered into a single RESPONSE ERROR message.

Although an UPDATE command would normally produce repeated Information Field responses for each request, a RESPONSE ERROR for an unsupported request will be sent only once.

42	RESPONSE ERROR
<count=variable>	Byte count (not including command and count).
<name>	Information Field name(s)

NOTES:

1. A Controller can be assured that, as a result of this message, every request for data will produce some kind of response under normal circumstances.
2. The following example presents a possible sequence of responses to a READ of three information fields, two of which are unsupported by the device ("BAD1" and "BAD2"), and the third of which is supported ("GOOD"):

Command:

F0 7F <device_ID> <mcc> <READ> <count=03> <BAD1> <BAD2> <GOOD> F7
Responses:

F0 7F <device_ID> <mcr> <RESPONSE ERROR> <count=01> <BAD1> F7

F0 7F <device_ID> <mcr> <RESPONSE ERROR> <count=01> <BAD2> F7

F0 7F <device_ID> <mcr> <GOOD> <..data..> F7

43 COMMAND ERROR [read only]

This response will be transmitted by the Controlled Device:

- (a) when requested by a READ command from the Controller;
- or (b) automatically, whenever an "enabled" error occurs.

Errors are "enabled" by the setting of the COMMAND ERROR LEVEL Information Field. If the error code of the newly detected error is less than or equal to the COMMAND ERROR LEVEL value, then that error is enabled. If greater than, the error is disabled.

```
43          COMMAND ERROR
<count=04+ext+count_1>
              Byte count
<flags>        Error flag bits: 0 gfedcba
                  a = Error Halt flag
                      0 = false (condition after a COMMAND ERROR
                           RESET, MMC RESET or power up)
                      1 = Error halt in effect:
                           Set by the occurrence of an "enabled" error.
                           All commands received after that which
                           caused the error will have been discarded;
                           No further commands will be processed
                           until a COMMAND ERROR RESET is
                           received.
                  b = PROCEDURE [ASSEMBLE] error flag:
                      0 = false
                      1 = Error found while pre-checking commands
                           embedded in a Procedure assembly.
                  c = EVENT [DEFINE] error flag:
                      0 = false
                      1 = Error found while pre-checking an embedded
                           Event command.
                  d = 0
                  e = Unsolicited COMMAND ERROR response flag:
                      0 = This transmission in response to a READ
                           request.
                      1 = This transmission unsolicited, and caused by
                           occurrence of an "enabled" error and the
                           consequent setting of the Error halt flag.
                  f = Previous COMMAND ERROR transmission flag:
                      0 = COMMAND ERROR field not transmitted since
                           the most recent error was recorded in it.
                           (Reset to 0 after each error occurrence.)
                      1 = COMMAND ERROR field (with most recent
                           error) has been transmitted previously.
                           (Set to 1 whenever COMMAND ERROR is
                           transmitted, whether unsolicited or not).
                  g = 0
<level>      Current setting of the COMMAND ERROR LEVEL Information Field.
<error>       Error code:  
          00 = reserved for extensions  
          01 thru 7E (see COMMAND ERROR CODE LIST below)  
          7F = No errors since power up or MMC RESET.
```

<i><count_1></i>	Byte count of following offset and command string. (For "Receive buffer overflow" and "Command Sysex length" errors, or if ee = 7F, set <i><count_1></i> = 00 and omit <i><offset></i> and <i><command string></i> .)
<i><offset></i>	Offset relative to the start of <i><command string></i> of the byte which caused the error (<i><offset=00></i> for first byte of string). Set to 7F if byte position is unavailable or undefined due to the nature of the error.
<i><command string></i>	Command which caused the most recent error. (Must be included in its entirety, with the exception that truncation may occur where the command is too large for the response Sysex, or where the length is uncertain due to the nature of the error.): <command name> or <command name> <coun> <command data>

COMMAND ERROR CODE LIST:

Error codes are classified in much the same way as are MMC commands and responses. Code 00 is reserved for extensions. Properties exhibited by any particular error class will be inherited by the corresponding extended class. For example, error codes 00 20 thru 00 3F will be classified as IMMEDIATE OPERATIONAL ERRORS, the same as codes 20 thru 3F.

Reaction to any "enabled" error within a Controlled Device is the same, whatever the error:

- Update the COMMAND ERROR Information Field;
- Set the "Error halt" flag;
- Automatically transmit the COMMAND ERROR field;
- Discard all commands received after that which caused the error;
- Discard and do not process any further commands until the "Error halt" flag has been reset (either by a COMMAND ERROR RESET or an MMC RESET).

Reaction to a "disabled" error depends on the error classification, and is in each case described below.

MAJOR ERRORS

- 01 = Receive buffer overflow
- 02 = Command Sysex length error (EOX or status byte not at legal message boundary)
- 03 = Command *<count>* error (inconsistent with Sysex length)
- 04 = Information Field *<count>* error during WRITE (inconsistent with Sysex length)
- 05 = Illegal Group name (7F)
- 06 = Illegal Procedure name (7F)
- 07 = Illegal Event name (7F)
- 08 = Illegal name extension beyond 2nd level
(i.e. 00 00 00 received where a "name" was expected)
- 09 = Segmentation Error (see COMMAND SEGMENT Command description)

Reaction to a "disabled" MAJOR ERROR is as follows:

- Update the COMMAND ERROR Information Field (do not set the "Error halt" flag);
- Discontinue all parsing of the current Sysex;
- Do not execute any command containing an error;
- Continue normal operations as soon as possible.

IMMEDIATE OPERATIONAL ERRORS

(Commands embedded within a PROCEDURE or an EVENT will not cause these errors until the procedure or event is actually executed.) *2

- 20 = UPDATE list overflow
- 21 = GROUP buffer overflow
- 22 = Undefined PROCEDURE
- 23 = PROCEDURE buffer overflow
- 24 = Undefined EVENT
- 25 = EVENT buffer overflow
- 26 = Blank time code

Reaction to a "disabled" IMMEDIATE OPERATIONAL ERROR is as follows:

Update the COMMAND ERROR Information Field (do not set the "Error halt" flag);
Continue parsing the current Sysex at the next message boundary;
Do not execute the command containing the error.

IMPLEMENTATION ERRORS

- 40 = Unsupported command
- 41 = Unrecognized sub-command
- 42 = Unrecognized command data
- 43 = Unsupported Information Field name in command data area
- 44 = Unsupported Information Field name in READ or UPDATE request within a PROCEDURE
- 45 = EVENT trigger source unavailable or unsupported
- 46 = Nested PROCEDURE [ASSEMBLE]
- 47 = Recursive PROCEDURE [EXECUTE]
- 48 = Nested EVENT [DEFINE]
- 49 = PROCEDURE [ASSEMBLE] within EVENT [DEFINE]

- 60 = Attempted WRITE to unsupported Information Field
- 61 = Attempted WRITE to Information Field which is "read only" (by definition or implementation)
- 62 = Unrecognized Information Field data during WRITE
- 63 = Unsupported Information Field name in Information Field data field during WRITE

Reaction to a "disabled" IMPLEMENTATION ERROR is as follows:

Update the COMMAND ERROR Information Field (do not set the "Error halt" flag);
Continue parsing the current Sysex at the next message boundary;
Do not execute the command containing the error;

If the error is found while pre-checking commands which are embedded in a Procedure or Event definition, then discard that definition, and continue parsing at the next message boundary AFTER the PROCEDURE [ASSEMBLE] or EVENT [DEFINE] command itself.

NOTES:

1. After power up or an MMC RESET, the COMMAND ERROR Information Field will assume the following state:

<count=04> <flags=00> <level=00> <error=7F> <count_1=00>

- *2. To clarify some PROCEDURE and EVENT error handling details, consider the following example in which an EVENT is defined within a PROCEDURE:

F0 7F <device_ID> <mcc>
<PROCEDURE> <count=0A> <[ASSEMBLE]> <procedure_name>
<EVENT> <count=06> <[DEFINE]> <event_name>
 <flags=00> <SELECTED TIME CODE> <GP6>
 <RECORD STROBE>

F7

- (a) If the procedure definition is longer than the available procedure memory, then a "PROCEDURE buffer overflow" error will be generated.
- (b) By comparison, even if the embedded EVENT definition would overflow available EVENT space, an "EVENT buffer overflow" will not occur when the procedure is defined, but may do so when the procedure is eventually executed.
- (c) If <procedure_name> contained 7Fhex, then an "Illegal Procedure name" error would be generated.
- (d) If <event_name> contained 7Fhex, then an "Illegal Event name" error would be generated, and the PROCEDURE [ASSEMBLE] error flag would be set, as the error occurred while pre-checking the EVENT command which is embedded within the procedure.
- (e) If register <GP6> were not supported by the device, then an "Unsupported Information Field name in command data area" error would be generated. In addition, both the PROCEDURE [ASSEMBLE] and EVENT [DEFINE] error flags would be set.

44 COMMAND ERROR LEVEL [read/write]

Command Errors are "enabled" by the setting of the COMMAND ERROR LEVEL Information Field. If the error code of a newly detected error is less than or equal to the COMMAND ERROR LEVEL value, then that error is enabled. If greater than, the error is disabled.

The default condition, after power up or an MMC RESET, is "All errors disabled".

44

<count=01>

vv

COMMAND ERROR LEVEL

Byte count.

Level:

00 = All errors disabled (Default)

01 thru 7E: Selective error enabling (refer to the
COMMAND ERROR Information Field description).

7F = All errors enabled

NOTES:

1. An extension set error code is compared on the basis of its final (non-zero) byte only.
2. When operating in an "open loop" configuration, it is recommended that errors be disabled.
3. COMMAND ERROR LEVEL will typically be used to enable errors according to their classification. For example, a level of 1F will enable all "Major" errors; 2F will enable "Major" and "Operational" errors; etc.
4. Refer also to the COMMAND ERROR Information Field and the COMMAND ERROR RESET Command descriptions.

45 TIME STANDARD [read/write]

Contains the nominal time code type to be used by the Controlled Device.
No default value is specified.

45 <count=01> <type>	TIME STANDARD Byte count. Frame count encoded as: 0 tt 00000 tt = time type: 00 = 24 frame 01 = 25 frame 10 = 30 drop frame 11 = 30 frame
----------------------------	---

NOTES:

1. For each of the MMC time code Information Fields, this nominal setting may be overridden by specific occurrences. For example, SELECTED TIME CODE will use the frame rate received by the time code reader; GENERATOR TIME CODE may be set to a different frame rate when a new time code value is loaded; etc.
Refer to Appendix B, "Time Code Status Implementation Tables" for usage of the tt (time type) bits which are embedded in each time code Information Field.
2. For a "clean" change of time standard, the following command sequence is recommended:
<MMC RESET> <TIME STANDARD> <count=01> <type>

46 SELECTED TIME CODE SOURCE [read/write]

Selects the source of time code to be presented in the SELECTED TIME CODE Information Field. Devices without access to time code should default to the "tape counter" selection. Otherwise, Longitudinal Time Code (LTC) should be used as the default.

46 <count=01> ss	SELECTED TIME CODE SOURCE Byte count. Source identification: 00 = Longitudinal Time Code (LTC) *1 *2 01 = Vertical Interval Time Code (VITC) *2 *3 02 = "Tape Counter" *2 04 = Auto VITC/LTC *2 *3 *4 7F = As defined locally [write only]
------------------------	--

NOTES:

- *1. Local implementations of Pilot Tone or Bi-Phase readers should present synthesized time code as Longitudinal Time Code (LTC).
- *2. LTC, VITC and Auto VITC/LTC may be updated by Tachometer or Control Track pulses in the absence of the selected code, for example, during fast wind modes.
- *3. Where VITC is not available, then default to LTC when either VITC or Auto VITC/LTC are selected.
- *4. Automatic VITC/LTC switchover characteristics are to be determined locally.

47 SELECTED TIME CODE USERBITS [read only]

Contains the userbits most recently extracted from the SELECTED TIME CODE.
The Information Field SELECTED TIME CODE SOURCE determines the source of these userbits.

47	SELECTED TIME CODE USERBITS
<count=09>	Byte count (not including command and count).
u1 thru u9	Standard Userbits Specification

48 MOTION CONTROL TALLY [read only]

Tallies: (a) the current "Motion Control State" of the Controlled Device,
and (b) specifies its success in achieving that state,
and (b) the current "Motion Control Process" of the Controlled Device,
and specifies its success at accomplishing that process.

Motion Control States and Processes are described in Section 3 "Standard Specifications".

48	MOTION CONTROL TALLY
<count=03+ext>	Byte count (not including command and count).
ms	Most recently activated Motion Control State: 00 = reserved for extensions 01 = STOP 02 = PLAY 04 = FAST FORWARD 05 = REWIND 09 = PAUSE 0A = EJECT 45 = VARIABLE PLAY 46 = SEARCH 47 = SHUTTLE 48 = STEP
mp	Most recently activated Motion Control Process: 00 = reserved for extensions 0B = CHASE 44 = LOCATE 7F = No MCP's currently active *1
ss	Status and success levels: 0 bbb 0 aaa aaa = MCS Success level (see below) bbb = MCP Success level (see below)

Valid "MCS Success levels" for the various MCS commands are:

STOP:	000 = Transition in progress 001 = Completely stopped 010 = Failure 011 = Deduced motion *2
-------	--

FAST FORWARD, REWIND, PLAY (unresolved):

000 = Transition in progress
001 = Requested motion achieved
010 = Failure
011 = Deduced motion *2

Resolved PLAY: **000 = Transition in progress**

001 = Playing and resolved (servo lock)
010 = Failure
101 = Playing but not resolved

PAUSE: **000 = Transition in progress**

001 = Completely stopped
010 = Failure

EJECT: **000 = Transition in progress**

001 = Media ejected/unloaded
010 = Failure

VARIABLE PLAY, SEARCH, SHUTTLE:

000 = Transition in progress
001 = Requested motion achieved
010 = Failure

STEP: **000 = Transition in progress**

001 = STEP completed, holding position
010 = Failure
100 = STEP in progress

Valid "MCP Success levels" for the various MCP commands are:

LOCATE: **000 = Actively locating**

001 = Locate complete - transport stopped at the requested locate point
010 = Failure
100 = Actively locating with Deferred Play pending *3
110 = Actively locating with Deferred Variable Play pending *4

CHASE: **000 = Actively attempting to synchronize in play mode**

001 = Successfully synchronized (play mode only)
010 = Failure
100 = Actively moving and chasing the master in non-play mode (typically high speed wind mode etc.)
110 = Parked *5

NOTES:

- *1. The MCP command tally must return to this inactive state when the current MCP has been terminated by receipt of an MCS command (with the exception that LOCATE will not be terminated by DEFERRED PLAY or DEFERRED VARIABLE PLAY). It will also be the condition after power up or an MMC RESET.
The "MCP Success level" must always be reset to 000 while there are "No MCP's currently active".

- *2. If a synchronizer or other interface is interposed between the Controller and the actual transport, then commands issued outside of that interface (for example by the operator at the transport itself) may not be directly monitored by the interface, but may be successfully deduced. The deduced motion will be reported in the MCS Command tally.
- *3. Refer to the DEFERRED PLAY command description.
- *4. Refer to the DEFERRED VARIABLE PLAY command description.
- *5. Parked status, valid only during the CHASE MCP, indicates that the slave (chasing) machine is stopped and ready to synchronize with the master device. This is useful when the master device has been LOCATE'd to a certain position, and the Controller must ascertain whether or not the slave has "caught up". The "stopped" condition is not sufficient in this case, as the slave may pass through that state while aligning itself with the master position.
- 6. MCS and MCP commands which cause "MAJOR" or "IMPLEMENTATION" COMMAND ERROR's must never appear in the "Most recently activated" bytes of this Information Field. (Refer to the COMMAND ERROR Information Field for definitions of these terms). On the other hand, an MCS or MCP command which encounters an "IMMEDIATE OPERATIONAL" COMMAND ERROR will appear in MOTION CONTROL TALLY, but with a "Success level" set to "Failure".
- 7. Any attempt to execute MCS or MCP commands when the most recent MCS command is EJECT will likely result in failure. Ejected media will typically require operator intervention before normal operation can be resumed.

49 VELOCITY TALLY [read only]

Tallies actual transport velocity at all times, and is independent of the prevailing Motion Control State and/or Process.

49 <i><count=03></i> <i>sh sm sl</i>	VELOCITY TALLY Byte count. Standard Speed Specification
--	---

4A STOP MODE [read/write]

Determines whether the Controlled Device should attempt to provide monitoring of recorded material while stopped as a result of a STOP command.

4A <i><count=01></i> <i>cc</i>	STOP MODE Byte count: Mode code: <code>00</code> = Disable monitoring. <code>01</code> = Enable monitoring.*1 <code>7F</code> = As defined locally [Default/write only]
--	--

NOTES:

- *1. With monitoring enabled, the STOP command is effectively converted to PAUSE, with the exception that there is no support for the "record-pause" mode, and that recording tracks will always exit from record.

2. A Controlled Device which cannot provide monitoring while stopped need not support STOP MODE.
3. STOP MODE affords a Controller the opportunity to apply the STOP command universally to all devices, assuming that STOP MODE has been set up in accordance with the operator's preferences.
4. Changing STOP MODE will not affect a stopped condition which has already been established.
5. STOP MODE governs all STOP commands received explicitly from the Controller. It does not apply to commands issued from the device's control panel. Neither does it apply to STOP commands issued automatically during the execution of a "Motion Control Process" (MCP), with the exception that any MCP which leaves the device in a stopped state at the end of its processing, must at that time honor the STOP MODE setting.
6. STOP MODE may be used to force VTR's to produce picture while stopped. This would also prevent cassette-style VTR's from "unthreading" at each STOP command.

4B FAST MODE [read/write]

Determines whether the Controlled Device should attempt to provide monitoring of recorded material during the execution of subsequent FAST FORWARD or REWIND commands from the Controller.

4B	FAST MODE
<count=01>	Byte count.
cc	Mode code:
	00 = Move at maximum velocity, without monitoring.
	01 = Move at maximum velocity attainable with monitoring of sufficient quality that recorded material is recognizable.
	7F = As defined locally [Default/write only]

NOTES:

1. A device need only support FAST MODE if the two monitoring alternatives are in fact available while fast motion is taking place.
2. Changing this field will not affect a FAST FORWARD or REWIND operation which is already in progress.
3. FAST MODE governs only those FAST FORWARD and REWIND commands received explicitly from the Controller. It does not apply to commands issued from the device's control panel, nor does it apply to FAST FORWARD and REWIND commands issued automatically during the execution of a "Motion Control Process" (MCP).
4. FAST MODE may be used to force VTR's to produce picture while winding tape. This would also prevent cassette-style VTR's from "unthreading" before initiating fast motion.
5. If a device outputs both picture and audio, then picture monitoring alone must follow the requirements of FAST MODE. Concurrent audio monitoring remains at the discretion of the equipment manufacturer.
6. FAST MODE should not be used to control an ATR's Tape Lifter mechanism unless audio outputs can be restricted to comfortable listening levels, requiring no external attenuation.
(Dynamic lifter control, without regard to output level, is provided elsewhere in MIDI Machine Control.)
ATR's without the capability of controlled monitoring at wind speeds should not support FAST MODE.

4C RECORD MODE [read/write]

Selects the mode of subsequent operation of the RECORD STROBE or RECORD STROBE VARIABLE commands. Changing this Field while tracks are already Recording [or Rehearsing] will not affect those tracks.

4C	RECORD MODE
<count=01>	Byte count.
dd	Mode:
	00 = Disabled
	01 = ATR:Record VTR:Insert *1
	02 = ATR:Record VTR:Assemble
	04 = Rehearse
	05 = ATR:Record VTR:Crash/Full Record
	7F = As selected locally [Default/write only]

"ATR:Record":

Record on all tracks specified by the TRACK RECORD READY Information Field.

"Rehearse":

All monitoring functions mimic those produced by the "01 ATR:Record/VTR:Insert" mode of record operation, without actually erasing old material or recording new. *2

"VTR:Insert":

Assumes that recording is to take place on a tape which has been pre-recorded with Control Track information.

The Control Track will be preserved during recording.

Clean and correctly timed transitions will be achieved between previously recorded material and the new material at both the record punch in and punch out locations.

Recording channels are determined by the TRACK RECORD READY Information Field.

"VTR:Assemble":

The Control Track and all program channels will be recorded upon. *3*4

A clean, correctly timed transition will be achieved between previously recorded material and new material at the record punch in location only, assuming that previously recorded material already exists at that location.

"VTR:Crash/Full Record":

Control Track and all program channels will be recorded upon. *3

No attempt will be made to achieve clean transitions or to match Control Track timing between previously recorded material and new material.

NOTES:

- *1. Most recording will be performed in the "01 ATR:Record/VTR:Insert" RECORD MODE.
- *2. On many devices, the delay between receipt of a RECORD STROBE command and the actual onset of rehearsing may be slightly different to the delay between RECORD STROBE and actual recording.
- *3. The TRACK RECORD READY Information Field, if supported, will not be affected by the selection of "VTR:Assemble" or "VTR:Crash/Full Record" modes. These modes will, however, override all TRACK RECORD READY settings, and force recording on all tracks plus the Control Track.
- *4. MMC does not currently support a "track selectable" VTR:Assemble mode.
- 5. VTR modes "Insert", "Assemble" and "Crash/Full Record" may be used by non-VTR devices which employ similar Control Track schemes.

4D RECORD STATUS [read only]

Reflects actual record and rehearse operations taking place at the Controlled Device.

4D RECORD STATUS
<count=01> Byte count.
ss Status: 0 d c b aaaa
 aaaa = Current Record/Rehearse activity (Hex digit):
 0 = No Record/Rehearse
 1 = ATR:Recording VTR:Insert Recording
 2 = VTR:Assemble Recording *1
 4 = Rehearsing
 5 = VTR:Crash/Full Record *1
 6 = Record-Pause
 b = Local Record inhibit flag (1 = inhibited) *2
 c = Local Rehearse inhibit flag (1 = inhibited) *2
 d = No Tracks Active (i.e recording or rehearsing); *3
 Negative-OR of all TRACK RECORD STATUS bits;
 Only valid when aaaa is non-zero.

NOTES:

- *1. While RECORD STATUS indicates "VTR:Assemble Recording" or "VTR:Crash/Full Record", the TRACK RECORD STATUS Information Field will indicate that all available tracks are recording.
- *2. "Local inhibit" flags may be set due to operator action or due to record tab orientation in cassettes or disks etc.
- *3. "No Tracks Active", when set to a 1, indicates that despite the record [rehearse] status shown in aaaa, no tracks are actually recording [rehearsing]. Whether or not this condition can arise is device-dependent. A Controller may choose to ignore this bit, or to interpret the condition as a "non-record" ["non-rehearse"].

4E TRACK RECORD STATUS [read only]

Contains bitmap of the tracks that are currently recording [or rehearsing]. Whether recording or rehearsing is tallied by the RECORD STATUS Information Field.

In all cases, the appropriate bit is set to 1 if the track is recording [rehearsing]. Unused bits must be zero. The Controlled Device need transmit only as many bytes of this response as are required. Tracks not included in a Response transmission will be assumed not to be recording [rehearsing]. A Byte count of 00 may be used if no tracks are recording [rehearsing].

4E TRACK RECORD STATUS
<count=variable> Byte count.
r0 r1 r2 . . . Standard Track Bitmap (see Section 3).

4F TRACK RECORD READY [read/write]

A "track" is moved into a "record ready" state when its bit is set to 1 in this track bitmap.

Upon receipt of the next RECORD STROBE or RECORD STROBE VARIABLE command, if recording or rehearsing is enabled in the RECORD MODE Information Field, tracks which are "record ready" but are

not recording [or rehearsing] will enter record [or rehearse], while tracks which are recording [rehearsing] but are not "record ready" will exit record [rehearse].

Changing this Information Field will not in itself cause tracks to enter or to exit record [or rehearse].

When read as a Response, the Controlled Device need transmit only as many bytes as are required. Tracks not included in a Response transmission will be assumed not to be in a TRACK RECORD READY state. A Byte count of 00 may be used if no tracks are in a ready state.

When written to by a WRITE command, tracks not included in the transmission will be set to the not ready state. A Byte count of 00 may be used if all tracks are to be disabled.

4F TRACK RECORD READY
<count=variable> Byte count of following bitmap.
r0 r1 r2 . . Standard Track Bitmap (see Section 3).

NOTES:

1. Before any recording or rehearsing can take place, Record or Rehearse must also be enabled in the RECORD MODE Information Field.
2. TRACK RECORD READY will not be affected by the selection of "VTR:Assemble" or "VTR:Crash/Full Record" in the RECORD MODE Information Field. These modes will, however, override all TRACK RECORD READY settings, and force recording on all tracks plus the Control Track.
3. The MASKED WRITE command may be used to change individual bits in the TRACK RECORD READY Information Field.

50 GLOBAL MONITOR [read/write]

Selects Playback or Input monitor modes for all tracks.

Two playback modes are defined:

- (a) "Synchronous" mode will be regarded as "normal" playback for a majority of devices. The term is derived from the fact that playback signals are synchronous with any new signals being recorded;
- (b) "Repro" mode is commonly implemented in ATR's by the use of a separate tape head optimized for playback as opposed to recording. The physical separation of the Repro head from the Record head causes the playback signal to be out of sync with signals being recorded.

GLOBAL MONITOR may be overridden on a track by track basis by the settings of the TRACK SYNC MONITOR, TRACK INPUT MONITOR and TRACK MUTE Information Fields.

50 GLOBAL MONITOR
<count=01> Byte count.
dd Mode:
 00 = Playback ["Synchronous"] (Default)
 01 = Input / Full EE
 02 = Playback ["Repro"] *2
 7F = As selected locally [write only]

NOTES:

1. Although many ATR's regard "Repro" as their native playback mode, the "Synchronous" mode must become the default mode when the device is addressed by MIDI Machine Control. This will provide a uniform approach for all Controlled Devices.

*2. If "Repro" mode is not supported, then use "Synchronous" playback.

51 RECORD MONITOR [read/write]

Selects the conditions under which track Inputs are to be monitored at their respective Outputs during Record operations.

Applies only to those tracks selected for "Synchronous" playback. Such selections are made either at the device's control panel or by writing to the GLOBAL MONITOR or TRACK SYNC MONITOR Information Fields, if supported.

A track designated for time code will not be affected by the RECORD MONITOR setting.

51 <count=01> dd	RECORD MONITOR Byte count. Mode: 00 = Record Only 01 = Record or Non-Play 02 = Record or Record-Ready 7F = As selected locally [Default/write only]
------------------------	---

"Record Only":

All tracks that are set to monitor Synchronous Playback will monitor Input, only when recording.

Upon the conclusion of a record operation, those tracks will revert back to Synchronous Playback.

"Record or Non-Play":

All tracks that are set to monitor Synchronous Playback will monitor input when recording. Upon the conclusion of a record operation, those tracks will revert back to Synchronous Playback. In addition, all Record Ready tracks will monitor Input when not in PLAY mode.

"Record or Record-Ready":

All tracks that are set to monitor Synchronous Playback, and are set to Record Ready or are Recording, will monitor Input.

NOTES:

1. RECORD MONITOR is typically applied to audio tracks only.
2. Reiteration, in table form, of the three RECORD MONITOR settings:

	Record Ready and Play	Record Ready and Non-play	RECORDING
00 Record Only:	Sync	Sync	Input
01 Record or Non-Play:	Sync	Input	Input
02 Record or Record-Ready:	Input	Input	Input
3. Actual monitoring may be overridden by the TRACK INPUT MONITOR and/or TRACK MUTE Information Fields.			

52 TRACK SYNC MONITOR [read/write]

Selects individual tracks that will present "Synchronous" playback on their respective outputs.
(Refer to the GLOBAL MONITOR Information Field for a description of "Synchronous" playback.)

TRACK SYNC MONITOR will always override the GLOBAL MONITOR setting for the tracks selected, but will itself be overridden by the TRACK INPUT MONITOR and TRACK MUTE Information Fields.

For any particular track, these overrides may be tabulated as follows (x="don't care"):

TRACK SYNC MONITOR bit	TRACK INPUT MONITOR bit	TRACK MUTE bit	Resultant monitoring :
0	0	0	As defined by GLOBAL MONITOR
1	0	0	"Synchronous" playback
x	1	0	Input
x	x	1	Mute

When read as a Response, the Controlled Device need transmit only as many bytes of TRACK SYNC MONITOR as are required. Tracks not included in a Response transmission will be assumed not to be individually selected for "Synchronous" playback. A Byte count of 00 may be used if no tracks are individually selected.

When written to by a WRITE command, tracks not included in the transmission will have their individual TRACK SYNC MONITOR bits reset to zero. A Byte count of 00 may be used if all tracks are to be reset.

52 TRACK SYNC MONITOR
<count=variable> Byte count of following bitmap.
r0 r1 r2 . . . Standard Track Bitmap (see Section 3).

NOTES:

1. TRACK SYNC MONITOR is intended as an adjunct to the "Repro" playback mode in the GLOBAL MONITOR Information Field, and allows combinations of tracks in "Repro" and tracks in "Sync" playback. This type of functionality is traditionally associated with ATR audio tracks.
2. The RECORD MONITOR Information Field will further govern "Synchronous" track monitoring during record operations.
3. The MASKED WRITE command may be used to change individual bits in the TRACK SYNC MONITOR Information Field.
4. A READ or UPDATE will return the TRACK SYNC MONITOR setting only, and may not reflect the final track monitoring configuration which results from the combination of GLOBAL MONITOR, TRACK SYNC MONITOR, TRACK INPUT MONITOR and TRACK MUTE.

53 TRACK INPUT MONITOR [read/write]

Selects individual tracks that will monitor Input signals at their respective Outputs.

TRACK INPUT MONITOR will always override both the TRACK SYNC MONITOR and GLOBAL MONITOR Information Field settings, but will itself be overridden by the TRACK MUTE Information Field. For any particular track, these overrides may be tabulated as follows (x="don't care"):

TRACK SYNC MONITOR bit	TRACK INPUT MONITOR bit	TRACK MUTE bit	Resultant monitoring :
0	0	0	As defined by GLOBAL MONITOR
1	0	0	"Synchronous" playback
x	1	0	Input
x	x	1	Mute

When read as a Response, the Controlled Device need transmit only as many bytes of TRACK INPUT MONITOR as are required. Tracks not included in a Response transmission will be assumed not to be selected for individual Input monitoring. A Byte count of 00 may be used if no tracks are individually selected.

When written to by a WRITE command, tracks not included in the transmission will have their individual TRACK INPUT MONITOR bits reset to zero. A Byte count of 00 may be used if all tracks are to be reset.

53 TRACK INPUT MONITOR
<count=variable> Byte count of following bitmap.
r0 r1 r2 . . . Standard Track Bitmap (see Section 3).

NOTES:

1. The MASKED WRITE command may be used to change individual bits in the TRACK INPUT MONITOR Information Field.
2. A READ or UPDATE will return the TRACK INPUT MONITOR setting only, and may not reflect the final track monitoring configuration which results from the combination of GLOBAL MONITOR, TRACK SYNC MONITOR, TRACK INPUT MONITOR and TRACK MUTE.

54 STEP LENGTH [read/write]

Defines the distance unit used by the STEP Command.

The STEP LENGTH is in turn measured in 1/100's of a time code frame.

54	STEP LENGTH
<count=01>	Byte count.
nn	Number of frames/100 in the STEP unit. Default value = 32hex (frame/2).

55 PLAY SPEED REFERENCE [read/write]

Determines whether a Controlled Device should control its speed internally when in standard PLAY mode, or allow direct play-speed control from an external source.

The contents of this field will be ignored during internal execution of CHASE or "Free Resolve" PLAY MODE, as play-speed is then under the control of the internal synchronization procedures.

55	PLAY SPEED REFERENCE
<count=01>	Byte count.
rr	Reference: 00 = Internal 01 = External 7F = As selected locally [Default/write only]

56 FIXED SPEED [read/write]

Selects a nominal fixed play-speed for a Controlled Device which supports more than one speed. All other velocity measurements will be calculated relative to this fixed speed.

56	FIXED SPEED
<count=01>	Byte count.
pp	Speed: 3F = next lower speed 40 = Medium, or "standard" speed 41 = next higher speed 7F = As selected locally [Default/write only]

NOTES:

1. If a WRITE command contains an out-of-range speed value, then the nearest available speed will be enabled.
2. Speed values must be assigned contiguously.
3. Examples:
 - (a) A three speed ATR might equate 3F, 40 and 41 with its Low, Medium and High speeds respectively. When written, all values in the range 00 thru 3F would in fact produce Low speed, and values 41 thru 7E would produce High speed.
 - (b) A VHS video deck may equate 40 with its normal speed, and provide 3F and 3E as alternative lower speeds for extended playing time.

57 LIFTER DEFEAT [read/write]

Defeats the tape lifter mechanism of a controlled reel-to-reel device, allowing tape contact with the heads.

57	LIFTER DEFEAT
<count=01>	Byte count.
cc	Control:
	00 = No defeat (Default)
	01 = Defeat
	7F = As selected locally [write only]

NOTE:

Many ATR's will produce excessive audio monitoring levels when lifters are defeated.

58 CONTROL DISABLE [read/write]

When disabled, the Controlled Device will ignore all Commands and all WRITE's involving the "Transport Control" (Ctrl) and "Synchronization" (Sync) message types, whether received directly from the Controller or as a result of Procedure or Event execution (see Section 4, Index List, for message type definitions). All other message types will remain active.

The Controller is thereby denied any direct control of the device itself, but may continue to monitor (READ) all Information Fields.

58	CONTROL DISABLE
<count=01>	Byte count.
cc	Control:
	00 = Enable (Default)
	01 = Disable
	7F = As selected locally [write only]

NOTES:

1. This Information Field will typically be implemented by synchronizers and other interfaces which may be interposed between the Controller and the target device. The effect then is that the synchronizer will disable its control outputs, leaving the target device totally free of external control.
2. The CONTROL DISABLE Information Field itself will not be disabled.

59 RESOLVED PLAY MODE [read/write]

Selects the manner in which the Controlled Device establishes its nominal fixed speed forward operation, when commanded by the PLAY command.

59	RESOLVED PLAY MODE
<count=01>	Byte count.
dd	Mode:
	00 = Normal, not resolved
	01 = Free Resolve Mode
	7F = As selected locally [Default/write only]

"Normal":

Achieve PLAY as defined by the PLAY SPEED REFERENCE. No relationship is implied to any time code or other frame reference.

"Free Resolve Mode":

Achieve PLAY in a manner that resolves the frame edges of the SELECTED TIME CODE to a locally defined Frame Reference, data independent, maintaining resolve data independent. *1

NOTES:

- *1. Future versions of the MIDI Machine Control specification may allow selection of this Frame Reference from, for example, a video reference signal, or simply the sync word of the incoming SELECTED MASTER CODE.
2. This Information Field need not be supported by devices which are inherently self-resolving (e.g. most VTR's).

5A CHASE MODE [read/write]

Selects the manner in which the Controlled Device achieves, and maintains synchronization between its SELECTED TIME CODE and the SELECTED MASTER CODE, when commanded by the CHASE command.

5A	CHASE MODE
<count=01>	Byte count.
dd	Mode:
	00 = Absolute Standard Mode
	01 = Absolute Resolve Mode
	7F = As selected locally [Default/write only]

"Absolute Standard Mode":

Achieve synchronism to the SELECTED MASTER CODE data dependent, maintain synchronism data dependent.

"Absolute Resolve Mode":

Achieve synchronism to the SELECTED MASTER CODE data dependent, maintain synchronism data independent.

NOTE:

Resolving, or locking "data independent", is simply a matter of synchronizing the SELECTED TIME CODE frame edges to an appropriate master Frame Reference. The actual numeric values of the time codes used while resolving are ignored. (Future versions of MIDI Machine Control may allow selection of this Frame Reference from, for example, a video reference signal, or simply the sync word of the incoming SELECTED MASTER CODE.)

5B GENERATOR COMMAND TALLY [read only]

Tallies the running state of time code generator

<i>5B</i> <i><count=02></i> <i>gg</i> <i>ss</i>	<p>GENERATOR COMMAND TALLY</p> <p>Byte count.</p> <p>Most recent generator command:</p> <p style="margin-left: 20px;">00 = Stop 01 = Run 02 = Copy/Jam</p> <p>Status (bit = 1 if status true) and success level: 0 0 cb 0 aaa</p> <p>aaa = Success level:</p> <p style="margin-left: 20px;">000 = Transition in progress 001 = Successful 010 = Failure</p> <p>b = Loss of Time Code Source data (during Copy/Jam) c = Loss of External Frame Sync Reference</p>
--	---

5C GENERATOR SET UP [read/write]

Controls the operating modes of the time code generator.

<i>5C</i> <i><count=03+ext></i> <i><reference></i> <i><source></i> <i><copy/jam></i>	<p>GENERATOR SET UP</p> <p>Byte count (not including command and count).</p> <p>Generator Frame Sync References: 0 yyyy 0 nnn</p> <p>nnn = Frame Sync Reference for Run mode:</p> <p style="margin-left: 20px;">000 = Internal crystal: "Standard" mode *1 001 = Locally defined external Frame Reference 010 = Internal crystal: "Drop A" mode *1 011 = Internal crystal: "Drop B" mode *1 111 = As locally defined [write only]</p> <p>yyy = Frame Sync Reference for Copy/Jam mode:</p> <p style="margin-left: 20px;">000 = Time Code Source frame edges 001 = Locally defined external Frame Reference 111 = As locally defined [write only]</p> <p>Time Code Source for Copy/Jam:</p> <p style="margin-left: 20px;">00 = reserved for extensions 01 = SELECTED TIME CODE (Default) 02 = SELECTED MASTER CODE 7F = As locally defined [write only]</p> <p>Copy/Jam mode:</p> <p style="margin-left: 20px;">00 = If the Time Code Source of the copied GENERATOR TIME CODE stops or disappears, then the GENERATOR TIME CODE should also stop. 01 = If the Time Code Source of the copied GENERATOR TIME CODE stops or disappears, then the GENERATOR TIME CODE should continue to run with no interruption in the number stream (also called <u>Time Code Jam mode</u>).</p>
--	---

NOTES:

*1 Internal Crystal Rate Table:

<i>nnn</i>	GENERATOR	TIME	CODE	time	type	(tt)
000 (Standard)	24	25	30DF	30		
010 (Drop A)	24	25	29.97	30		
011 (Drop B)	23.976	24.975	29.97	29.97		

2. Future versions of MMC may provide support for generator color framing.

5D GENERATOR USERBITS [read/write]

Contains the current userbit contents being generated by the time code generator.

5D GENERATOR USERBITS
<count=09> Byte count.
u1 thru u9 Standard Userbits Specification

5E MIDI TIME CODE COMMAND TALLY [read only]

Tallies the running state of MIDI Time Code generator.

5E MIDI TIME CODE COMMAND TALLY
<count=02> Byte count.
mm Most recent MIDI time code command:
 00 = Off
 02 = Follow time code
ss Status: 0 0000 aaa
 aaa = Success level:
 000 = Transition in progress
 001 = Successful
 010 = Failure

5F MIDI TIME CODE SET UP [read/write]

Controls the operating modes of the MIDI time code generator.

5F MIDI TIME CODE SET UP
<count=02+ext> Byte count (not including command and count).
<flags> MTC flags: *0 gfedcba*
 a = "Transmit while stopped" flag:
 0 = Inhibit MTC transmission when Time Code
 Source is detected to have stopped
 1 = Continue transmission when source has stopped,
 with data type specified by bit *b*.
 b = "Stopped" data type, if enabled by bit *a*:
 0 = Quarter frame messages with no frame
 incrementing.
 1 = Full Messages transmitted at regular, but
 unspecified, intervals.
 c = "Transmit while fast" flag:
 0 = Inhibit MTC transmission when Time Code
 Source is detected be moving at a rate
 higher than at least twice its nominal frame
 rate.
 1 = When source is moving faster than at least twice
 its nominal frame rate, continue
 transmission with data type specified by bit
 d.
 d = "Fast" data type, if enabled by bit *c*:
 0 = Quarter frame messages. *1
 1 = Full Messages transmitted at regular, but
 unspecified, intervals.
 e = "Transmit userbits" flag:
 0 = Inhibit MTC transmission of userbits.
 1 = Transmit UserBits Message whenever userbits
 contents change, or at regular, unspecified,
 intervals.
 f = MMC Response cable mute flag:
 0 = MIDI Time Code, when operating, will be
 transmitted on the MMC Response cable
 1 = MIDI Time Code will not be transmitted on the
 MMC Response cable, but may appear at
 other, unspecified, MIDI Out ports.
 g = 0
<source> Time Code Source:
 00 = reserved for extensions
 01 = SELECTED TIME CODE
 02 = SELECTED MASTER CODE
 06 = GENERATOR TIME CODE
 07 = MIDI TIME CODE INPUT
 (produces a "soft-THRU" mode)
 7F = As locally defined [write only]

NOTES:

- *1 In order to adequately track a high speed Time Code Source, and to guarantee correct MTC reception, these quarter frame messages should:

- (a) run at the nominal frame rate;
 - and (b) be transmitted in "bursts", where each "burst" consists of several time code frames incrementing in a normal frame sequence.
2. The "time type" flags (*tt*) of the transmitted MIDI Time Code will be the same as those of the Time Code Source.

60 PROCEDURE RESPONSE [read only]

Allows the Controller to read back any, or all, of the assembled Procedures. The name of the Procedure to be read back must first be established by the PROCEDURE [SET] command.

If the PROCEDURE [SET] command specified "set all PROCEDURES", then a separate PROCEDURE RESPONSE will be transmitted for each Procedure currently assembled within the Controlled Device. See also the PROCEDURE Command.

60	PROCEDURE RESPONSE
<count=variable>	Byte count (not including command and count).
<procedure>	Procedure Name in the range 00 thru 7E. 7F = invalid Procedure set (i.e either the PROCEDURE [SET] command was never issued, or it specified an undefined Procedure, or there are currently no Procedures defined.) No further data required (<count=01>).
<command #1..>	
<command #2..>	
<command #3..>	

61 EVENT RESPONSE [read only]

Allows the Controller to read back any, or all, of the defined Events. The name of the Event to be read back must first be established by the EVENT [SET] command.

If the EVENT [SET] command specified "set all EVENT's", then a separate EVENT RESPONSE will be transmitted for each Event currently defined within the Controlled Device.

See also the EVENT Command.

61	EVENT RESPONSE
<count=variable>	Byte count of following bytes.
<event>	Event Name in the range 00 thru 7E. 7F = invalid Event set (i.e either the EVENT [SET] command was never issued, or it specified an undefined Event, or there are currently no Events defined.) No further data required (<count=01>).
<flags>	Event control flags (see the EVENT [DEFINE] command for bit definitions.)
<trigger source>	Information Field name of time code stream relative to which the event is to be triggered (see the EVENT [DEFINE] command).
<i>hr mn sc fr ff</i>	Event time (type <i>{ff}</i>).
<command..>	Single command plus data.

62 TRACK MUTE [read/write]

Selects individual tracks that will have their Output signals muted.

TRACK MUTE overrides all other monitoring selections.

When read as a Response, the Controlled Device need transmit only as many bytes of TRACK MUTE as are required. Tracks not included in a Response transmission will be assumed not to be selected for individual muting. A Byte count of 00 may be used if no tracks are muted.

When written to by a WRITE command, tracks not included in the transmission will have their individual TRACK MUTE bits reset to zero (track unmuted). A Byte count of 00 may be used if all tracks are to be unmuted.

62 TRACK MUTE
<count=variable> Byte count of following bitmap.
r0 r1 r2 . . . Standard Track Bitmap (see Section 3). *2

NOTES:

1. The MASKED WRITE command may be used to change individual bits in the TRACK MUTE Information Field.
- *2. TRACK MUTE is directed primarily at audio tracks. The "Video" bit will normally be zero.

63 VITC INSERT ENABLE [read/write]

Selects whether or not Vertical Interval Time Code is to be embedded in the video signal which is received at the Controlled Device's video input. If that video is subsequently to be recorded, then the VITC information will be recorded along with it.

VITC is derived from the Device's time code generator, and is therefore controlled also by the GENERATOR COMMAND and the GENERATOR SET UP Information Field. *1

63 VITC INSERT ENABLE
<count=03> Byte count.
cc Control:
 00 = Disable
 01 = Enable
 7F = As selected locally [Default/write only]
h1 First horizontal line number for VITC insertion;
 0Ah hex thru 12hex NTSC
 06hex thru 14hex PAL
 7F = As selected locally [Default/write only]
h2 Second (non-adjacent) horizontal line number for VITC insertion,
 where h2 > h1+1.
 0Ch hex thru 14hex NTSC
 08hex thru 16hex PAL
 7F = As selected locally [Default/write only]

NOTES:

- *1. The <reference> data in the GENERATOR SET UP Information Field may be internally overridden when VITC is enabled, as the generator must then be referenced to video.

64 RESPONSE SEGMENT [no access]

Allows a response (or a string of responses), which is greater in length than the maximum MMC System Exclusive data field length (48 bytes), to be divided into segments and transmitted piece by piece across multiple System Exclusives.

Responses received by the Controller in this way will be treated exactly as if they had arrived all in the same sysex.

RESPONSE SEGMENT must always be the first response in its sysex, and there must be no other responses in the sysex save those which are contained within the body of RESPONSE SEGMENT itself.

Segment divisions need not fall on response boundaries. Partial responses, which may occur at the end of a RESPONSE SEGMENT sysex, must be detected by the Controller so that response processing may be correctly resumed when the next segment arrives.

With the exception of WAIT or RESUME messages, if a non-segmented (i.e. normal) sysex is received by a Controller when a "subsequent" segment sysex from the same Controlled Device was expected, it will be processed normally, and de-segmentation for that device will be cancelled. *1

Refer to Section 2 "General Structure" - "Segmentation" for further explanation and examples.

64	RESPONSE SEGMENT
<count=variable>	Byte count (response string segment length + 1)
si	Segment Identification: 0 f ssssss
	f: 1 = first segment
	0 = subsequent segment
	sssssss = segment number (down count, last=000000)
<.. responses ..>	Response string segment

NOTE:

- *1. A Controller must maintain separate de-segmentation processes for each of the Controlled Devices which are attached to it.

65 FAILURE [no access]

Warns of a catastrophic failure of the Controlled Device i.e. a failure which requires local operator intervention.

65	FAILURE
<count>	Byte count (<count=0> if no data).
<data . . >	ASCII data for optional display at the Controller (may be truncated to fit display size).

7C WAIT [no access]

The WAIT Response signals the Controller that the Controlled Device's receive buffer is filling (or that the Device is otherwise busy), and that Machine Control Command transmissions must be discontinued until receipt of a RESUME from the Controlled Device. Any Command transmission which is currently in progress will be allowed to proceed up to its normal End of System Exclusive (F7). Transmission of subsequent Commands may resume after receipt of a RESUME from the Controlled Device. The Commands WAIT and RESUME, however, are not inhibited by the WAIT Response. Neither is transmission of the WAIT Response itself inhibited by receipt of a WAIT Command.

A Controller must guarantee:

- (i) to recognize the receipt of a WAIT message within 10 milliseconds after the arrival of the End of System Exclusive (F7) of that WAIT message;
- and (ii) to then halt all transmissions at the next available MMC System Exclusive boundary (up to 53 bytes, the maximum MMC sysex length, may therefore have to be transmitted before the halt can take effect).

The WAIT Response is transmitted as the only response in its Sysex

i.e. F0 7F <device_ID> <mcr> <WAIT> F7.

7C

WAIT

NOTES:

1. Correct operation of the WAIT response requires a certain minimum size for the MIDI receive buffer in the Controlled Device. Refer to Appendix E, "Determination of Receive Buffer Size".
2. Additional WAIT responses may be transmitted by a Controlled Device should its receive buffer continue to fill.

7F RESUME [no access]

Signal to the Controller that the Controlled Device is ready to receive Machine Control Commands. The default (power up) state is "ready to receive".

The RESUME Response is used primarily to allow the Controller to resume transmissions after a WAIT. Transmission of the RESUME Response is not inhibited by the receipt of a WAIT Command.

The RESUME Response is transmitted as the only response in its Sysex

i.e. F0 7F <device_ID> <mcr> <RESUME> F7.

7F

RESUME

Appendix A EXAMPLES

EXAMPLE 1

A very basic tape transport has been manufactured which supports the Commands: <STOP> <DEFERRED PLAY> <FAST FORWARD> <REWIND> <RECORD STROBE> <RECORD EXIT> <MMC RESET> <WRITE> <LOCATE> <MOVE>; and the Information Fields: <SELECTED TIME CODE> <GP0/LOCATE POINT>.

The manufacturer has published the device's SIGNATURE: 01 00 00 00

0C
7B 41 00 00 00 00 00 00 00 00 00 00
11 20
02
02 02

Communication is in the "open loop" mode only. The transport accepts commands at its MIDI In port, but provides no responses at its MIDI Out. The machine's device_ID is dipswitch selectable, and has been set to 01hex.

The following command sequence is typical of "open loop" style MMC operation:

Play:

F0 7F 01 <mcc> <DEFERRED PLAY> F7

Stop:

F0 7F 01 <mcc> <STOP> F7

Reset 'counter' to all zeroes, 30 frame:

F0 7F 01 <mcc> <WRITE> <count=06> <SELECTED TIME CODE> 60 00 00 20 00 F7

Fast forward:

F0 7F 01 <mcc> <FAST FORWARD> F7

Stop:

F0 7F 01 <mcc> <STOP> F7

Establish a locate point:

F0 7F 01 <mcc> <MOVE> <count=02> <GP0/LOCATE POINT> <SELECTED TIME CODE> F7

Play:

F0 7F 01 <mcc> <DEFERRED PLAY> F7

Punch into record:

F0 7F 01 <mcc> <RECORD STROBE> F7

Punch out of record:

F0 7F 01 <mcc> <RECORD EXIT> F7

Return to locate point and play:

F0 7F 01 <mcc>
<LOCATE> <count=02> <[I/F]> <GP0/LOCATE POINT> <DEFERRED PLAY>
F7

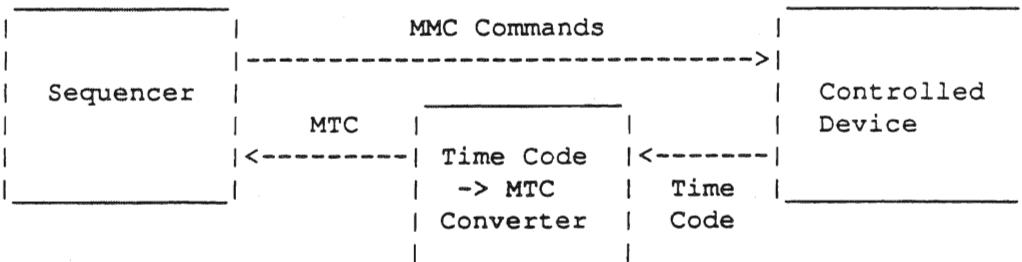
Return to Zero:

F0 7F 01 <mcc> <LOCATE> <count=06> <[TARGET]> 60 00 00 00 00 F7

EXAMPLE 2

MIDI Machine Control and MIDI Time Code are combined in this two part example. Control of a single device by a computer based Sequencer will typically follow one of these formats.

Example 2A: Open Loop MMC with External Time Code to MTC Converter:



The unusual feature of this hook up is that the Sequencer knows the exact tape time code position of the Controlled Device (via MTC), whereas the device itself does not. In most cases, the device will rely on tachometer pulses to update its internal <*SELECTED TIME CODE*> register.

MMC communication is once again in the "open loop" mode only. The transport is identical to that of Example 1, and accepts commands at its MIDI In port while providing no responses at its MIDI Out. Its device_ID has been set to 01hex. Commands from the Sequencer to the device will be shown towards the left side of the page, and MIDI Time Code from the Converter towards the right. The Sequencer is assumed to be operated using a mouse and keyboard.

The Sequencer always issues an MMC Reset at the beginning of the session:

F0 7F 01 <mcc> <MMC RESET> F7

The operator clicks the "Remote Machine PLAY" button on the Sequencer screen. As the Sequencer supports the convention that this button may also be used to punch out of record, it transmits the following message:

F0 7F 01 <mcc> <RECORD EXIT> <DEFERRED PLAY> F7

The Sequencer begins interpreting MIDI Time Code. After a short stabilization period, frame 01:02:03:04 (30 frames/sec, non-drop) is received:

F1 04 . . F1 10 . . F1 23 . . F1 30 . .
F1 42 . . F1 50 . . F1 61 . . F1 76 . .
F1 06 . .

Immediately after receiving the frames digit of the next two-frame MTC "word", the Sequencer forces the current MTC time back into the Controlled Device's time code register:

F0 7F 01 <mcc> <WRITE> <count=06>
<*SELECTED TIME CODE*> 61 02 03 26 00
F7

The operator clicks "Remote Machine RECORD". The Sequencer internally saves the current MTC time (01:02:13:20), and sends the command:

F0 7F 01 <mcc> <RECORD STROBE> F7

Operator clicks "Remote Machine PLAY" to punch out of record:

F0 7F 01 <mcc> <RECORD EXIT> <DEFERRED PLAY> F7

Operator clicks "Remote Machine STOP":

F0 7F 01 <mcc> <STOP> F7

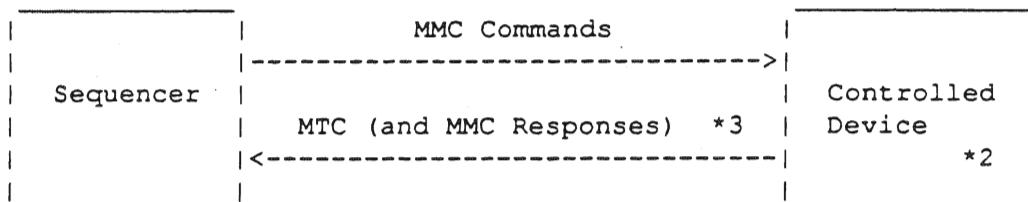
Operator requests that the Sequencer review the material recorded on the Remote Machine. The Sequencer locates the device to 01:02:08:20, five seconds prior to the previously stored record punch in point:

F0 7F 01 <mcc>
<LOCATE> <count=06> <[TARGET]> 61 02 08 14 00
<DEFERRED PLAY>
F7

NOTES:

1. There will usually be some drift between MIDI Time Code and the tachometer updated values maintained by the device. The extent of this drift will depend not only on the mechanical condition of the device, but also on the possibility that the time code has been recorded slightly "off-speed".
In order to minimize the effect of the drift, the Sequencer may, at regular intervals, force the most recently received MIDI Time Code back into the device's <SELECTED TIME CODE> register. Care should be exercised so that only valid and current time code values are used.
2. Despite taking the above precaution, some positioning errors during execution of the LOCATE command may be unavoidable.
3. Using the EVENT command to punch in and out of record may be problematic when the device is operating from tach pulses only.

Example 2B: Closed Loop MMC with Internal Time Code to MTC Conversion:



This example represents a considerable improvement over example 2A. Here the device itself performs the time code to MTC conversion, and therefore has access to exact time code positioning information.

Commands from the controller to the device will be shown towards the left side of the page, and Responses and MIDI Time Code from the device towards the right. The Controlled Device conforms to Guideline Minimum Set #3, with the addition of MIDI Time Code command and information fields. Its device_ID is shown as <ident>.

The Sequencer always issues an MMC Reset at the beginning of the session:

F0 7F <ident> <mcc> <MMC RESET> F7

It then checks the device's capabilities:

F0 7F <ident> <mcc> <READ> <count=01> <SIGNATURE> F7

The machine replies:

```
F0 7F <ident> <mcr>
  <SIGNATURE> <count=2E> 01 00 00 00
  <count_1=14>
    7F 61 00 00 00 00 00 00 00 00 00
    7F 70 7F 00 00 00 00 00 00 00 09
  <count_2=14>
    02 1E 00 00 00 02 1E 00 00 00
    3F 62 07 01 0C 37 00 00 00 09
```

F7

The Sequencer sets the Command Error Level for "Major" and "Immediate Operational" errors only, and sets up the MIDI Time Code generator/converter:

```
F0 7F <ident> <mcc>
  <WRITE> <count=03>
    <COMMAND ERROR LEVEL> <count=01> 3F
  <WRITE> <count=04>
    <MIDI TIME CODE SET UP> <count=02>
      00 <SELECTED TIME CODE>
```

F7

The operator requests Play. The Sequencer also requests that MIDI Time Code be turned on:

```
F0 7F <ident> <mcc>
  <RECORD EXIT> <PLAY>
  <MIDI TIME CODE COMMAND> <count=01> 02
```

F7

MIDI Time Code 03:02:20:28 (drop frame) arrives at the Sequencer's MIDI In:

```
F1 0C . . F1 11 . . F1 24 . . F1 31 . .
F1 42 . . F1 50 . . F1 63 . . F1 74 . .
```

The Sequencer checks that the device's internal copy of the current time code is in fact the same as the received MTC:

```
F0 7F <ident> <mcc>
  <READ> <count=01> <SELECTED TIME CODE>
```

F7

The device responds in the middle of the MTC frame 03:02:21:00-01:

```
F1 00 . . F1 10 . . F1 25 . . F1 31 . .
F1 42 . .
F0 7F <ident> <mcr>
  <SELECTED TIME CODE> 43 02 15 21 00
F7
  . . F1 50 . . F1 63 . . F1 74 . .
```

The operator "marks" on the fly a record punch in time of 03:02:27:08, which the Sequencer saves internally.

Some time later, the operator marks a record out point of 03:02:41:15, which the Sequencer also saves.

The operator then requests that the Sequencer rewind the transport and put it into record between the marked punch in and punch out points. Recording is to occur on tracks 1 and 2 only.

The Sequencer initiates a locate action, with MIDI Time Code turned off, and monitors MOTION CONTROL TALLY until the locate is complete:

```
F0 7F <ident> <mcc>
  <MIDI TIME CODE COMMAND> <count=01> 00
  <LOCATE> <count=06> <[TARGET]> 43 02 16 08 00
  <UPDATE> <count=02>
    <[BEGIN]> <MOTION CONTROL TALLY>
```

F7

The device responds immediately, showing that locating has begun in the reverse (rewind) direction:

```
F0 7F <ident> <mcr>
  <MOTION CONTROL TALLY> <count=03>
    <REWIND> <LOCATE> 01
```

F7

The device may pass through several different stages in the course of the locate:

```
F0 7F <ident> <mcr>
  <MOTION CONTROL TALLY> <count=03>
    <FAST FORWARD> <LOCATE> 00
```

F7

```
.
.
.
F0 7F <ident> <mcr>
  <MOTION CONTROL TALLY> <count=03>
    <FAST FORWARD> <LOCATE> 01
```

F7

```
.
.
.
F0 7F <ident> <mcr>
  <MOTION CONTROL TALLY> <count=03>
    <STOP> <LOCATE> 00
```

F7

```
.
.
.
F0 7F <ident> <mcr>
  <MOTION CONTROL TALLY> <count=03>
    <STOP> <LOCATE> 11
```

F7

After receiving the above "Locate complete" tally, the Sequencer ceases monitoring MOTION CONTROL TALLY, and then checks that the device has indeed located to the correct position:

```
F0 7F <ident> <mcc>
  <UPDATE> <count=02> <[END]> 7F
  <READ> <count=01> <SELECTED TIME CODE>
```

F7

The device's response, although representing a locate error of three frames, is deemed by the Sequencer to be satisfactory.

```
F0 7F <ident> <mcr>
    <SELECTED TIME CODE> 43 02 16 25 00
F7
```

The Sequencer now sets up all subsequent record activities, and chooses to monitor the RECORD STATUS information field in order to update its screen display:

```
F0 7F <ident> <mcc>
<WRITE> <count=0F>
    <TRACK RECORD READY> <count=01> 60
    <GP1> 43 02 1B 08 00
    <GP2> 43 02 29 0F 00
<EVENT> <count=06> <[DEFINE]> <event#=01>
    <flags=00> <SELECTED TIME CODE> <GP1>
    <RECORD STROBE>
<EVENT> <count=06> <[DEFINE]> <event#=02>
    <flags=00> <SELECTED TIME CODE> <GP2>
    <RECORD EXIT>
<UPDATE> <count=02> <[BEGIN]> <RECORD STATUS>
F7
```

The device immediately returns RECORD STATUS ("not recording"):

```
F0 7F <ident> <mcr> <RECORD STATUS> <count=01> 00 F7
```

Finally, a Play command is issued, with MIDI Time Code turned on:

```
F0 7F <ident> <mcc>
<PLAY>
<MIDI TIME CODE COMMAND> <count=01> 02
F7
```

At the record punch in point (03:02:27:08), the device inserts "recording" status between MTC messages: *1

```
F1 08 . .
F0 7F <ident> <mcr>
    <RECORD STATUS> <count=01> 01
F7
. . F1 10 . . F1 2B . . F1 31 . .
F1 42 . . F1 50 . . F1 63 . . F1 74 . .
```

Similarly, "not recording" is returned at the punch out point (03:02:41:15): *1

```
F1 0E . . F1 10 . . F1 29 . . F1 31 . .
F1 42 . .
F0 7F <ident> <mcr>
    <RECORD STATUS> <count=01> 00
F7
. . F1 50 . . F1 63 . . F1 74 . .
```

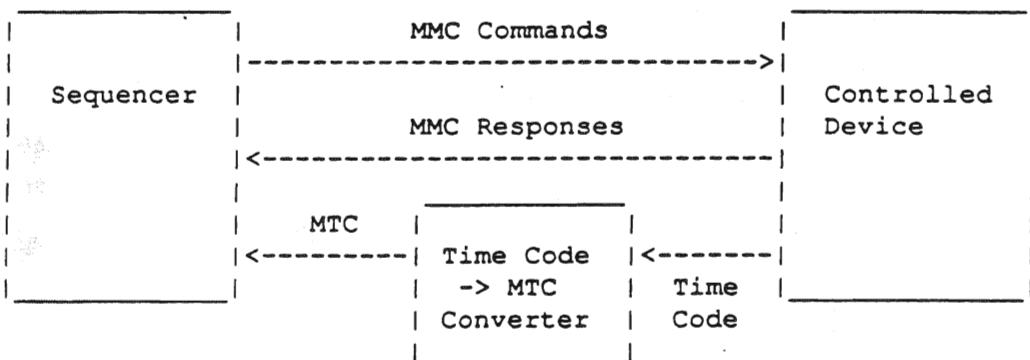
At the conclusion of recording, the transport is stopped, MIDI Time Code is turned off, as is monitoring of RECORD STATUS, and all tracks are returned to the "record not ready" state:

```
F0 7F <ident> <mcc>
<STOP>
<MIDI TIME CODE COMMAND> <count=01> 00
<UPDATE> <count=02> <[END]> 7F
<WRITE> <count=02>
<TRACK RECORD READY> <count=00>
```

F7

NOTES:

- *1 Although it is essentially the responsibility of the Controller (Sequencer) to keep the MIDI response line free of traffic during those periods when MTC timing is critical, an intelligent Controlled Device will make every attempt to prevent MMC Responses from causing jitter in the MTC messages.
- *2 Whole systems of machines could be operated using this configuration. An intelligent translator would be required which would represent such a system as a single virtual machine.
- *3 An additional MIDI In port at the Sequencer would make possible operations similar to this example while using an external Time Code to MTC converter. Control over MTC generation would be lost however, and the device would still need to be equipped with an internal time code reader.



EXAMPLE 3

This example represents a far more complex situation than the previous two. The controlled devices are two identical synchronizers (or tape transports with embedded synchronizers), and the controller is capable of initiating some basic automated "edit" sequences, as well as displaying time code and other transport related status. Communications are "closed loop". The controller's MIDI Out is fed to the MIDI In of both devices, using one device's MIDI Thru. The MIDI Outputs from the two devices are fed to separate MIDI Inputs at the controller. One of the devices has been designated as the "Master" for synchronization purposes, and its device_ID has been set to 01hex. The other device, the "Slave", has a device_ID of 02hex. All actions associated with assigning this Master, and the associated routing of "Master Time Code" to the Slave synchronizer, are assumed to have taken place at the devices themselves, and are not in this instance the concern of the MIDI system.

Each device supports the following Commands:

```
<STOP> <PLAY> <DEFERRED PLAY> <FAST FORWARD> <REWIND> <RECORD STROBE>
<RECORD EXIT> <CHASE> <COMMAND ERROR RESET> <MMC RESET> <WRITE> <READ>
<UPDATE> <LOCATE> <VARIABLE PLAY> <MOVE> <ADD> <SUBTRACT> <DROP FRAME ADJUST>
<PROCEDURE> <EVENT> <GROUP> <COMMAND SEGMENT> <DEFERRED VARIABLE PLAY> <WAIT>
<RESUME>
```

Each device supports the following Responses/Information Fields:

```
<SELECTED TIME CODE> <SELECTED MASTER CODE> <REQUESTED OFFSET> <ACTUAL OFFSET>
<LOCK DEVIATION> <GP0/LOCATE POINT> <GP1> <GP2> <GP3> <Short time codes>
<SIGNATURE> <UPDATE RATE> <RESPONSE ERROR> <COMMAND ERROR>
<COMMAND ERROR LEVEL> <TIME STANDARD> <MOTION CONTROL TALLY> <RECORD MODE>
<RECORD STATUS> <CONTROL DISABLE> <RESOLVED PLAY MODE> <CHASE MODE>
<PROCEDURE RESPONSE> <EVENT RESPONSE> <RESPONSE SEGMENT> <FAILURE> <WAIT>
<RESUME>
```

The published SIGNATURE for each device is as follows:

```
01 00 00 00
14
7F 71 00 00 00 00 00 00 00 00 00 00
3D 60 7F 00 00 00 00 00 00 00 00 09
14
3E 1E 00 00 00 3E 1E 00 00 00
3F 62 00 38 00 33 00 00 00 09
```

Commands from the controller to the devices will be shown towards the left side of the page, and Responses from the devices towards the right. Master and Slave Responses are distinguished by their respective device_ID's (third byte of the Sysex).

The control methods shown here are constructed mainly to demonstrate the power and flexibility of MIDI Machine Control, and certainly do not represent the only, or even the best, approach to the task.

The Controller first clears all previous settings, and establishes a group consisting of both the master and slave:

```
F0 7F <all-call=7F> <mcc>
<MMC RESET>
<GROUP> <count=04> <[ASSIGN]> <group=7C> 01 02
```

F7

The Controller sets a 30 frame time standard in both devices, and enables all command errors:

```
F0 7F <group=7C> <mcc>
<WRITE> <count=06>
<TIME STANDARD> <count=01> 03
<COMMAND ERROR LEVEL> <count=01> <all=7F>
F7
```

The operator initiates a Play on the Master and then on the Slave.

(Note that this is not yet a CHASE operation):

```
F0 7F <master=01> <mcc> <PLAY> F7
F0 7F <slave=02> <mcc> <PLAY> F7
```

The Controller requests continuous updating of both time codes and motion tallies from both devices:

```
F0 7F <group=7C> <mcc>
<UPDATE> <count=03> <[BEGIN]>
<SELECTED TIME CODE>
<MOTION CONTROL TALLY>
F7
```

Both master and slave respond with full 5 byte time code and tallies:

Master is PLAYing at 00:22:05:12 (00:16:05:0C hex).

Slave is PLAYing at 10:01:58:28 (0A:01:3A:1C hex).

```
F0 7F <master=01> <mcr>
<SELECTED TIME CODE> 60 16 05 2C 00
<MOTION CONTROL TALLY> <count=03> <PLAY> 7F 01
F7
```

```
F0 7F <slave=02> <mcr>
<SELECTED TIME CODE> 6A 01 3A 3C 00
<MOTION CONTROL TALLY> <count=03> <PLAY> 7F 01
F7
```

The next time code from the master (00:22:05:13) is in "short" form, as only the frames have changed. There has been no change at all in the MOTION CONTROL TALLY:

```
F0 7F <master=01> <mcr>
<Short SELECTED TIME CODE> 2D 00
F7
```

Next slave code (10:01:58:29):

```
F0 7F <slave=02> <mcr>
<Short SELECTED TIME CODE> 3D 00
F7
```

More master (00:22:05:14) and slave (10:01:59:00) times . . notice that the slave has seen a change in its "seconds", and has transmitted the entire 5 bytes of time code:

```
F0 7F <master=01> <mcr>
<Short SELECTED TIME CODE> 2E 00
F7
```

```
F0 7F <slave=02> <mcr>
    <SELECTED TIME CODE> 6A 01 3B 20 00
F7
```

If the Controller's buffer were now filling up, it would transmit a WAIT request:

```
F0 7F <all-call=7F> <mcc> <WAIT> F7
```

(all transmissions halted)

Buffer clear:

```
F0 7F <all-call=7F> <mcc> <RESUME> F7
```

More master (00:22:05:16) and slave (10:01:59:02) time codes:

```
F0 7F <master=01> <mcr>
    <Short SELECTED TIME CODE> 30 00
F7
```

```
F0 7F <slave=02> <mcr>
    <Short SELECTED TIME CODE> 22 00
F7
```

Operator Stops the master:

```
F0 7F <master=01> <mcc> <STOP> F7
```

The master tally changes to reflect a "stopped" condition:

```
F0 7F <master=01> <mcr>
    <MOTION CONTROL TALLY> <count=03> <STOP> 7F 01
F7
```

More slave time code (10:01:59:03):

```
F0 7F <slave=02> <mcr>
    <Short SELECTED TIME CODE> 23 00
F7
```

Slave at 10:01:59:04:

```
F0 7F <slave=02> <mcr>
    <Short SELECTED TIME CODE> 24 00
F7
```

Operator Stops the slave:

```
F0 7F <slave=02> <mcc> <STOP> F7
```

The slave tally shows "stopped":

```
F0 7F <slave=02> <mcr>
    <MOTION CONTROL TALLY> <count=03> <STOP> 7F 01
F7
```

Operator requests that the current difference between the master and slave positions become the slave's synchronization offset:

```
F0 7F <slave=02> <mcc>
    <MOVE> <count=02>
        <REQUESTED OFFSET> <ACTUAL OFFSET>
F7
```

The Controller reads back the offset for display purposes:

```
F0 7F <slave=02> <mcc>
    <READ> <count=01> <REQUESTED OFFSET>
F7
```

The slave responds with an offset of 09:39:53:18.00 (09:27:35:12.00 hex):

```
F0 7F <slave=02> <mcr>
    <REQUESTED OFFSET> 69 27 35 12 00
F7
```

Operator pushes the "slave chase" button:

```
F0 7F <slave=02> <mcc> <CHASE> F7
```

The slave tally adjusts, showing chase mode, "stopped" and "parked" status:

```
F0 7F <slave=02> <mcr>
    <MOTION CONTROL TALLY> <count=03> <STOP> <CHASE> 61
F7
```

Operator Plays the master (slave follows):

```
F0 7F <master=01> <mcc> <PLAY> F7
```

Master (00:22:05:24) and slave (10:01:59:09) return new tallies and time codes.
The slave begins its synchronization process:

```
F0 7F <master=01> <mcr>
    <Short SELECTED TIME CODE> 38 00
    <MOTION CONTROL TALLY> <count=03> <PLAY> 7F 01
F7
```

```
F0 7F <slave=02> <mcr>
    <Short SELECTED TIME CODE> 29 00
    <MOTION CONTROL TALLY> <count=03> <PLAY> <CHASE> 01
F7
```

The operator chooses to observe the slave lock deviation (error) while synchronization is taking place:

```
F0 7F <slave=02> <mcc>
    <UPDATE> <count=02> <[BEGIN]> <LOCK DEVIATION>
F7
```

The slave obliges (deviation = +5.02 frames):

```
F0 7F <slave=02> <mcr>
    <LOCK DEVIATION> 60 00 00 05 02
F7
```

More master time (00:22:05:25):

```
F0 7F <master=01> <mcr>
    <Short SELECTED TIME CODE> 39 00
F7
```

Slave time (10:01:59:10) and deviation (-1.17 frames), not yet synchronized:

```
F0 7F <slave=02> <mcr>
    <Short SELECTED TIME CODE> 2A 00
    <Short LOCK DEVIATION> 41 17
F7
```

Slave time (10:02:01:00), deviation and tally (synchronization achieved):

```
F0 7F <slave=02> <mcr>
    <SELECTED TIME CODE> 6A 02 01 20 00
    <Short LOCK DEVIATION> 00 00
    <MOTION CONTROL TALLY> <count=03> <PLAY> <CHASE> 11
F7
```

With the Master in play, and the Slave synchronized, the operator now wishes to establish a small automatic "edit" by marking a Record "In" point and a Record "Out" point. The actual recording will be performed on the Slave machine.

From this point on, we shall only show significant UPDATE Responses.

Operator marks an "IN" point. The Controller captures the current time code by moving it into the GP1 general purpose register in both the master and slave machines using the group device_ID:

```
F0 7F <group=7C> <mcc>
    <MOVE> <count=02> <GP1> <SELECTED TIME CODE>
F7
```

Some time later, the operator marks an "OUT" point, and the Controller captures to the GP2 register:

```
F0 7F <group=7C> <mcc>
    <MOVE> <count=02> <GP2> <SELECTED TIME CODE>
F7
```

The operator now requests that the "edit" be performed automatically.

The Controller sets up events in the slave to punch into record at the time in GP1, and to punch out at time GP2. Both events are to be deleted after being triggered, and both are to be triggered only by forward motion play-speed time code.

The Controller also sets the slave's record mode and requests an automatic update of record status:

```
F0 7F <slave=02> <mcc>
  <EVENT> <count=06> <[DEFINE]> <event=#01>
    <flags=00> <SELECTED TIME CODE> <GP1>
    <RECORD STROBE>
  <EVENT> <count=06> <[DEFINE]> <event=#02>
    <flags=00> <SELECTED TIME CODE> <GP2>
    <RECORD EXIT>
  <WRITE> <count=03> <RECORD MODE> <count=01> 01
  <UPDATE> <count=02> <[BEGIN]> <RECORD STATUS>
```

F7

The slave immediately returns its current (non) record status:

```
F0 7F <slave=02> <mcr>
  <RECORD STATUS> <count=01> 00
F7
```

Next, the Controller sets up the master to (a) locate to a "preroll" point 5 seconds before the record punch in point in GP1 (GP0 is employed for the calculation); and (b) trigger an event to cause an automatic stop when the master reaches a point 2 seconds beyond the punch out point in GP2 (the trigger point is set up in GP3).

```
F0 7F <master=01> <mcc>
  <WRITE> <count=06> <GP0/LOCATE POINT> 60 00 05 00 00
  <SUBTRACT> <count=03> <GP0/LOCATE POINT> <GP1> <GP0/LOCATE POINT>
  <LOCATE> <count=02> <[I/F]> <GP0/LOCATE POINT>
  <WRITE> <count=06> <GP3> 60 00 02 00 00
  <ADD> <count=03> <GP3> <GP3> <GP2>
  <EVENT> <count=06> <[DEFINE]> <event=#01>
    <flags=10> <SELECTED TIME CODE> <GP3>
    <STOP>
```

F7

In addition to time codes and slave lock deviation, the following master and slave motion tallies will have been returned, indicating that the master is locating with the slave chasing:

```
F0 7F <master=01> <mcr>
  <MOTION CONTROL TALLY> <count=03>
    <REWIND> <LOCATE> 01
F7
```

```
F0 7F <slave=02> <mcr>
  <MOTION CONTROL TALLY> <count=03>
    <REWIND> <CHASE> 41
F7
```

.

.

.

Eventually the master will finish locating, but the slave may still be active:

```
F0 7F <master=01> <mcr>
    <MOTION CONTROL TALLY> <count=03>
        <STOP> <LOCATE> 11
F7

F0 7F <slave=02> <mcr>
    <MOTION CONTROL TALLY> <count=03>
        <REWIND> <CHASE> 41
F7
```

Finally, the slave parks with the master:

```
F0 7F <slave=02> <mcr>
    <MOTION CONTROL TALLY> <count=03>
        <STOP> <CHASE> 61
F7
```

Upon detecting that the machines have successfully located and parked, the Controller issues a play to the master, thus initiating the automated series of events previously loaded:

```
F0 7F <master=01> <mcc> <PLAY> F7
```

Master plays, slave attempts to synchronize:

```
F0 7F <master=01> <mcr>
    <MOTION CONTROL TALLY> <count=03> <PLAY> 7F 01
F7
```

```
F0 7F <slave=02> <mcr>
    <MOTION CONTROL TALLY> <count=03> <PLAY> <CHASE> 01
F7
```

Eventually, the slave is synchronized:

```
F0 7F <slave=02> <mcr>
    <MOTION CONTROL TALLY> <count=03> <PLAY> <CHASE> 11
F7
```

The slave punches into record at the pre-arranged point (and deletes Event #01):

```
F0 7F <slave=02> <mcr>
    <RECORD STATUS> <count=01> 01
F7
```

Some time later, the slave punches out (and deletes Event #02):

```
F0 7F <slave=02> <mcr>
    <RECORD STATUS> <count=01> 00
F7
```

Two seconds after the punch out, the master stops, causing the slave also to stop and park:

```
F0 7F <master=01> <mcr>
    <MOTION CONTROL TALLY> <count=03> <STOP> 7F 01
F7

F0 7F <slave=02> <mcr>
    <MOTION CONTROL TALLY> <count=03> <STOP> <CHASE> 61
F7
```

At the end of the session, the Controller mutes all update responses and disables both machines:

```
F0 7F <group=7C> <mcc>
    <UPDATE> <count=02> <[END]> <all=7F>
    <WRITE> <count=03> <CONTROL DISABLE> <count=01> 01
F7
```

Finally, the operator attempts to manipulate a (non-existent) time code generator in the master device:

```
F0 7F <master=01> <mcc>
    <READ> <count=01> <GENERATOR TIME CODE>
    <GENERATOR SET UP> <count=03>
        00 <SELECTED TIME CODE> 01
    <GENERATOR COMMAND> <count=01> 01
F7
```

The master first responds to the READ of the unsupported GENERATOR TIME CODE:

```
F0 7F <master=01> <mcr>
    <RESPONSE ERROR> <count=01> <GENERATOR TIME CODE>
F7
```

Then follows notification that the GENERATOR SET UP command is also unsupported. Since all errors have been enabled by the Controller, the master will enter "Error Halt" mode, and cease all command processing. The final command, GENERATOR COMMAND, will be discarded.

```
F0 7F <master=01> <mcr>
    <COMMAND ERROR> <count=0A>
        <flags=11> <level=7F> <error=40>
        <count_1=06> <offset=00>
    <GENERATOR SET UP> <count=03>
        00 <SELECTED TIME CODE> 01
F7
```

The Controller acknowledges the COMMAND ERROR message, thus enabling the master to resume command processing:

```
F0 7F <master=01> <mcc> <COMMAND ERROR RESET> F7
```

Appendix B TIME CODE STATUS IMPLEMENTATION TABLES

Time code status bits are defined in Section 3, "Standard Specifications". Their exact implementation for each MMC time code Information Field is shown in this Appendix.

01 SELECTED TIME CODE [read/write]:

Time code Status Bits		Value after power up or MMC RESET	Value during Normal Operations	Interpretation of time code bits contained in WRITE data
tt (time type)	TIME STANDARD or other internal default	TIME STAN-DARD or other internal default	If $n = 1$ (time code never read): $tt =$ TIME STANDARD (or default) or as loaded with WRITE or "Math" commands If $n = 0$: $tt =$ As read from time code	WRITE to tt only if bit $n = 1$ (time code never read), else ignore tt in WRITE data.
c (color frame)	0		If $n = 1$: $c = 0$ If $n = 0$: $c =$ As read from time code	Ignore c in WRITE data
k (blank)	1		$k = 0$ only after time code has been: (a) read (from "tape") or (b) incremented by tach pulses or (c) loaded by WRITE or (d) loaded by "Math" command Otherwise, $k = 1$	Always set $k = 0$ after a WRITE; Ignore k in WRITE data.
g (sign)	0	0		Ignore g in WRITE data.
i (final byte id)	1	1		Ignore i in WRITE data.
e (estimated code)	0		$e = 1$ only if <u>most recent</u> time code change came as a result of tachometer or control track pulse updating.	Always set $e = 0$ after a WRITE; Ignore e in WRITE data.
v (invalid code)	0	Set as required.		Always set $v = 0$ after a WRITE; Ignore v in WRITE data.
d (video field 1)	0 (0/1 if implemented)	Set/reset as required, and if implemented.		Ignore d in WRITE data.
n (no time code)	1	$n = 0$ <u>only after time code has been read from "tape"</u> .		Always set $n = 1$ after a WRITE; (i.e. "reset" to "time code never read"); Ignore n in WRITE data.

02 SELECTED MASTER CODE [read only]:

Same as SELECTED TIME CODE, with the exception that WRITE's cannot occur.

03 REQUESTED OFFSET [read/write]:

Time code <u>Status Bits</u>	Value after power up or MMC <u>RESET</u>	Value during Normal Operations	Interpretation of time code bits contained in WRITE data
tt (time type)	See next column.	Follows tt in SELECTED TIME CODE, except that REQUESTED OFFSET will remain <u>non-drop-frame</u> ($tt=11$) when SELECTED TIME CODE is drop-frame ($tt=10$).	Ignore tt in WRITE data. (Future versions of MMC may allow some variation here.)
c (color frame)	0	0	Ignore c in WRITE data.
k (blank)	1	$k = 0$ only after time code has been: (a) loaded by WRITE or (b) loaded by "Math" command Otherwise, $k = 1$	Always set $k = 0$ after a WRITE; Ignore k in WRITE data.
g (sign)	0	0/1 as required	OK to WRITE g bit.
i (final byte id)	0	0	If $i = 0$ in WRITE data: Load final data byte as subframes; If $i = 1$ in WRITE data: Ignore final data byte; Load subframes = 00

04 **ACTUAL OFFSET [read only]:**
05 **LOCK DEVIATION [read only]:**

		Value after power up or MMC
<u>Time code</u>	<u>RESET</u>	<u>Value during Normal Operations</u>
<i>tt</i> (time type)	See next column.	Follows <i>tt</i> in SELECTED TIME CODE, except that ACTUAL OFFSET and LOCK DEVIATION will remain <u>non-drop-frame</u> (<i>tt=11</i>) when SELECTED TIME CODE is drop- frame (<i>tt=10</i>).
<i>c</i> (color frame)	0	0
<i>k</i> (blank)	0	0
<i>g</i> (sign)	0	0/1 as required
<i>i</i> (final byte id)	0	0

06 GENERATOR TIME CODE [read/write]:

<u>Time code Status Bits</u>	<u>Value after power up or MMC RESET</u>	<u>Value during Normal Operations</u>	<u>Interpretation of time code bits contained in WRITE data</u>
<i>tt</i> (time type)	TIME STANDARD or other internal default	<i>tt</i> = TIME STANDARD (or default), or as loaded by WRITE or "Math" commands or by a Copy/Jam from another time code source.	OK to WRITE to <i>tt</i> . Subsequently determines the time code counting mode of the Generator.
<i>c</i> (color frame)	0	0	Ignore <i>c</i> in WRITE data.
<i>k</i> (blank)	0	0	Ignore <i>k</i> in WRITE data.
<i>g</i> (sign)	0	0	Ignore <i>g</i> in WRITE data.
<i>i</i> (final byte id)	1	1	Ignore <i>i</i> in WRITE data.
<i>e</i> (estimated code)	0	0	Ignore <i>e</i> in WRITE data.
<i>v</i> (invalid code)	0	0	Ignore <i>v</i> in WRITE data.
<i>d</i> (video field 1)	0 (0/1 if implemented)	Set/reset as required, and if implemented.	Ignore <i>d</i> in WRITE data.
<i>n</i> (no time code)	0	0	Ignore <i>n</i> in WRITE data.

07 MIDI TIME CODE INPUT [read only]:

Time code <u>Status Bits</u>	Value after power up or MMC	<u>Value during Normal Operations</u>
<i>tt</i> (time type)	TIME STAN- DARD or other internal default	If <i>n</i> = 1 (time code never read): <i>tt</i> = TIME STANDARD (or default) If <i>n</i> = 0: <i>tt</i> = As read from MTC
<i>c</i> (color frame)	0	0
<i>k</i> (blank)	1	<i>k</i> = 0 only after MTC has been received at the device's MIDI In. Otherwise, <i>k</i> = 1
<i>g</i> (sign)	0	0
<i>i</i> (final byte id)	1	1
<i>e</i> (estimated code)	0	0
<i>v</i> (invalid code)	0	Set as required.
<i>d</i> (video field 1)	0	0
<i>n</i> (no time code)	1	<i>n</i> = 0 only after MTC has been received at the device's MIDI In. Otherwise, <i>n</i> = 1 (i.e. <i>n</i> = <i>k</i>)

08

thru

0F GP0 thru GP7 [read/write]:

<u>Time code Status Bits</u>	<u>Value after power up or MMC RESET</u>	<u>Value during Normal Operations</u>	<u>Interpretation of time code bits contained in WRITE data</u>
<i>tt</i> (time type)	TIME STAN-DARD or other internal default	<i>tt</i> = As loaded with WRITE or "Math" commands	OK to WRITE to <i>tt</i> .
<i>c</i> (color frame)	0	<i>c</i> = As loaded with WRITE or "Math" commands	OK to WRITE to <i>c</i> .
<i>k</i> (blank)	1	<i>k</i> = 0 only after time code has been: (a) loaded by WRITE or (b) loaded by "Math" command Otherwise, <i>k</i> = 1	Always set <i>k</i> = 0 after a WRITE; Ignore <i>k</i> in WRITE data.
<i>g</i> (sign)	0	<i>g</i> = As loaded with WRITE or "Math" commands	OK to WRITE to <i>g</i> .
<i>i</i> (final byte id)	0	0	If <i>i</i> = 0 in WRITE data: Load final data byte as subframes; If <i>i</i> = 1 in WRITE data: Ignore final data byte; Load subframes = 00

Appendix C SIGNATURE TABLE

COMMAND BITMAP ARRAY:

<u>Byte</u>	<u>Bit 6 (40h)</u>	<u>Bit 5 (20h)</u>	<u>Bit 4 (10h)</u>	<u>Bit 3 (08h)</u>	<u>Bit 2 (04h)</u>	<u>Bit 1 (02h)</u>	<u>Bit 0 (01h)</u>
c0	06 RECORD STROBE	05 REWIND	04 FAST FORWARD	03 DEFERRED PLAY	02 PLAY	01 STOP	00 (reserved)
c1	0D MMC RESET	0C COMMAND ERROR RESET	0B CHASE	0A EJECT	09 PAUSE	08 RECORD PAUSE	07 RECORD EXIT
c2	14	13	12	11	10	0F	0E
c3	1B	1A	19	18	17	16	15
c4	-	-	-	1F	1E	1D	1C
c5	26	25	24	23	22	21	20
c6	2D	2C	2B	2A	29	28	27
c7	34	33	32	31	30	2F	2E
c8	3B	3A	39	38	37	36	35
c9	-	-	-	3F	3E	3D	3C
c10	46 SEARCH	45 VARIABLE PLAY	44 LOCATE	43 UPDATE	42 READ	41 MASKED WRITE	40 WRITE
c11	4D ADD	4C MOVE	4B MIDI TIME CODE COMMAND	4A GENERATOR COMMAND	49 ASSIGN SYSTEM MASTER	48 STEP	47 SHUTTLE
c12	54 DEFERRED VARIABLE PLAY	53 COMMAND SEGMENT	52 GROUP	51 EVENT	50 PROCEDURE	4F DROP FRAME ADJUST	4E SUBTRACT
c13	5B	5A	59	58	57	56	55 RECORD STROBE VARIABLE
c14	-	-	-	5F	5E	5D	5C
c15	66	65	64	63	62	61	60
c16	6D	6C	6B	6A	69	68	67
c17	74	73	72	71	70	6F	6E
c18	7B	7A	79	78	77	76	75
c19	-	-	-	7F RESUME	7E	7D	7C WAIT

RESPONSE/INFORMATION FIELD BITMAP ARRAY:

<u>Byte</u>	<u>Bit 6 (40h)</u>	<u>Bit 5 (20h)</u>	<u>Bit 4 (10h)</u>	<u>Bit 3 (08h)</u>	<u>Bit 2 (04h)</u>	<u>Bit 1 (02h)</u>	<u>Bit 0 (01h)</u>
r0	06 GENERATOR TIME CODE	05 LOCK DEVIATION	04 ACTUAL OFFSET	03 REQUESTED OFFSET	02 SELECTED MASTER CODE	01 SELECTED TIME CODE	00 (reserved)
r1	0D GP5	0C GP4	0B GP3	0A GP2	09 GP1	08 GP0 / LOCATE POINT	07 MIDI TIME CODE INPUT
r2	14	13	12	11	10	0F GP7	0E GP6
r3	1B	1A	19	18	17	16	15
r4	-	-	-	1F	1E	1D	1C
r5	26 Short GENERATOR TIME CODE	25 Short LOCK DEVIATION	24 Short ACTUAL OFFSET	23 Short REQUESTED OFFSET	22 Short SELECTED MASTER CODE	21 Short SELECTED TIME CODE	20 (reserved)
r6	2D Short GP5	2C Short GP4	2B Short GP3	2A Short GP2	29 Short GP1	28 Short GP0 / LOCATE POINT	27 Short MIDI TIME CODE INPUT
r7	34	33	32	31	30	2F Short GP7	2E Short GP6
r8	3B	3A	39	38	37	36	35
r9	-	-	-	3F	3E	3D	3C
r10	46 SELECTED TIME CODE SOURCE	45 TIME STANDARD	44 COMMAND ERROR LEVEL	43 COMMAND ERROR	42 RESPONSE ERROR	41 UPDATE RATE	40 SIGNATURE
r11	4D RECORD STATUS	4C RECORD MODE	4B FAST MODE	4A STOP MODE	49 VELOCITY TALLY	48 MOTION CONTROL TALLY	47 SELECTED TIME CODE USERBITS
r12	54 STEP LENGTH	53 TRACK INPUT MONITOR	52 TRACK SYNC MONITOR	51 RECORD MONITOR	50 GLOBAL MONITOR	4F TRACK RECORD READY	4E TRACK RECORD STATUS

<u>Byte</u>	<u>Bit 6 (40h)</u>	<u>Bit 5 (20h)</u>	<u>Bit 4 (10h)</u>	<u>Bit 3 (08h)</u>	<u>Bit 2 (04h)</u>	<u>Bit 1 (02h)</u>	<u>Bit 0 (01h)</u>
r13	5B GENERATOR COMMAND TALLY	5A CHASE MODE	59 RESOLVED PLAY MODE	58 CONTROL DISABLE	57 LIFTER DEFEAT	56 FIXED SPEED	55 PLAY SPEED REFERENCE
r14	-	-	-	5F MIDI TIME CODE SET UP	5E MIDI TIME CODE COMMAND TALLY	5D GENERATOR USERBITS	5C GENERATOR SET UP
r15	66	65 FAILURE	64 RESPONSE SEGMENT	63 VITC INSERT ENABLE	62 TRACK MUTE	61 EVENT RESPONSE	60 PROCEDURE RESPONSE
r16	6D	6C	6B	6A	69	68	67
r17	74	73	72	71	70	6F	6E
r18	7B	7A	79	78	77	76	75
r19	-	-	-	7F RESUME	7E	7D	7C WAIT

Appendix D MIDI MACHINE CONTROL and MTC CUEING

It is anticipated that some devices will implement both MIDI Machine Control and MTC Cueing. In such cases, it may be expedient to merge the MMC Events with the MTC Cueing Event List.

In an effort to be both self-contained and adaptable to ESbus methods, MIDI Machine Control has not adopted MTC Cueing as its means of triggering events.

This section will attempt to define those areas where MIDI Machine Control and MTC Cueing overlap, as well as those where they do not.

Comparison of MIDI Machine Control and MTC Cueing event specifications:

MIDI Machine Control :

MMC "Events" trigger other MMC commands only.

Events are defined by the EVENT command.

Each event is unique, and is identified by a single 7-bit name.

Each Event definition specifies the source of the time code stream against which the Event should be triggered.

Events contain additional flags for triggering at other than play speeds, and when moving forward or reverse or either.

For compatibility with ESbus, each event may optionally be deleted after being triggered.

No overall event enable/disable is provided.

Error trapping checks for "Illegal Event name", "Undefined EVENT", "EVENT buffer overflow", and performs complete pre-checking of the command which is to be executed at the EVENT trigger time.

MTC Cueing :

MTC "Events" trigger event sequences, cues, and track punch in and out.

Events are defined by MTC Set-Up messages.

The same event number can be triggered at different times, and each Event List item is therefore identified by a combination of trigger time and event number.

No time code source is specified, but is assumed to be MIDI Time Code.

No motion variations or restrictions are specified.

Events are not deleted when triggering occurs.

Event enable/disable commands turn all events on and off without affecting the Event List itself.

No error trapping is provided.

Review of MTC Cueing messages, and their relationship to MMC:

- 01 **Punch In points**
- 02 **Punch Out points**
- 03 **Delete Punch In point**
- 04 **Delete Punch Out point**

These MTC messages are functionally duplicated by MIDI Machine Control commands, although exact translation is awkward.

The following example illustrates a translation of the MTC Punch In message into MMC commands:

MTC Cueing: *F0 7E <chan> 04 01 hr mn sc fr ff s1 sm F7*
 (where *s1 sm*= track number)

MMC: *F0 7F <device_ID> <mcc>*
 <PROCEDURE> <count=09> <[ASSEMBLE]> <procedure_name>
 <MASKED WRITE> <count=04>
 <TRACK RECORD READY> <byte #> <mask> <data=7F>
 <RECORD STROBE>
 <WRITE> <count=06> <GP0> *hr mn sc fr ff*
 <EVENT> <count=09> <[DEFINE]> <event_name>
 <flags=40> <MIDI TIME CODE INPUT> <GP0>
 <PROCEDURE> <count=02> <[EXECUTE]> <procedure_name>
 F7

- 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**
- 0E **Event Name in additional info.**

The MTC Event Start and Stop messages "imply that a large sequence of events or a continuous event is to be started or stopped" (MIDI 1.00 Detailed Specification).

MIDI Machine Control has no real equivalent for this style of Event.

- 0B **Cue points**
- 0C **Cue points with additional info.**
- 0D **Delete Cue point**

An MTC Cue Point "refers to individual event occurrences, such as marking 'hit' points for sound effects, reference points for editing, and so on" (MIDI 1.00 Detailed Specification).

Although closer in style to the MMC EVENT, these MTC Cue points would not appear to be intended for the lower level execution of specific machine commands. The "additional info" area certainly provides a cumbersome method for defining such commands.

We shall therefore regard MTC Cue points as representing an activity for which MMC has no exact equivalent.

00 : 00 00 Time Code Offset

The MTC Cueing Time Code Offset message may be translated literally into the corresponding MMC command:

MTC Cueing: F0 7E <chan> 04 00 hr mn sc fr ff 00 00 F7

MMC: F0 7F <device_ID> <mcc>
<WRITE> <count=06> <REQUESTED OFFSET> hr mn sc fr ff F7

00 : 01 00 Enable Event List
00 : 02 00 Disable Event List

MIDI Machine Control does not currently support such commands for its own EVENT's. When received, these messages should be applied only to those events defined by MTC messages, and should have no effect on MMC EVENT's at all.

00 : 03 00 Clear Event List

MIDI Machine Control provides its own methods for deleting events. When received, this message should also be applied only to those events defined by MTC messages, and should have no effect on MMC EVENT's at all.

00 : 04 00 System Stop

The MTC Cueing System Stop message may be translated into an MMC Event command as follows:

MTC Cueing: F0 7E <chan> 04 00 hr mn sc fr ff 04 00 F7

MMC: F0 7F <device_ID> <mcc>
<WRITE> <count=06> <GP0> hr mn sc fr ff
<EVENT> <count=06> <[DEFINE]> <event_name>
<flags=50> <SELECTED TIME CODE> <GP0>
<STOP>

F7

00 : 05 00 Event List Request

MIDI Machine Control EVENT's may be read back using the EVENT RESPONSE Information Field. Event List Request should cause transmission of only those events defined by MTC messages. MMC EVENT's should not be included.

Appendix E DETERMINATION OF RECEIVE BUFFER SIZE

Satisfactory operation of the WAIT handshake requires a certain minimum size for the MIDI receive buffer in an MMC device. The fact that a MIDI System Exclusive should not be interrupted once its transmission has begun can cause delays not only in the dispatching of a WAIT message, but also in the cessation of transmissions in response to a WAIT. A receiving device must provide enough buffer space so that buffer overflow will not occur should the device be receiving full bandwidth MIDI data between the time that it decides to transmit a WAIT and the time that the received data stream actually comes to a halt.

Complicating matters further, the use of external MIDI merging devices can considerably lengthen WAIT delays, or, at the very least, make them less predictable.

We shall first examine receive buffer requirements when external merging is not used. All calculations assume "worst case" scenarios.

Operation of WAIT in a Simple "Closed Loop":

The following diagram represents receive and transmit activity at an MMC device "A". We shall assume that another device, "B", is transmitting at full MIDI bandwidth, but that device "A" may transmit at a slower rate. Our objective is to determine the worst case minimum receive buffer size for device "A", when connected in a basic, point-to-point, "closed loop" configuration.

Rx:	<--sysex-1-->	<--sysex-2-->	<--sysex-3-->	<--sysex-4-->	<--sysex-5-->			
	*							
Tx:	<--sysex- $(S_A$ bytes)-->					<WAIT>		
Notes:	1	2		3	4	5	6	7
Times:	t_A		t_{SA}	t_{PA}	t_{WA}	t_B	t_{SB}	

1. A received byte causes device "A"'s receive buffer to fill beyond a predetermined threshold $\{H_A\}$, measured in bytes (see point "*" in the diagram).
2. After a delay, $\{t_A\}$, device "A" detects that its receive threshold has been crossed. In this worst case example, device "A"'s transmitter section has just begun transmitting a sysex of length $\{S_A\}$ bytes. Since transmission of a MIDI sysex cannot be interrupted once begun, it will be necessary to wait time $\{t_{SA}\}$ for the completion of this sysex before a WAIT message can be sent.
3. At the completion of the sysex, there may be a short delay, $\{t_{PA}\}$, while device "A" prepares the WAIT message.
4. Device "A" sends the WAIT message, which is itself 6 bytes long, and which will take time $\{t_{WA}\}$ to transmit.
5. WAIT message completes.

6. Meanwhile, over at device "B", there will be a delay, $\{t_B\}$, before the arrival of "A"'s WAIT message is detected. As before, worst case conditions dictate that device "B" has just begun transmission of an entirely new sysex, and cannot cease transmitting until after the EOX.
7. After a further sysex length delay, $\{t_{SB}\}$, device "B" finally halts transmissions.

The maximum time, $\{t_{max}\}$, from the crossing of "A"'s receive buffer threshold to the end of "B"'s sysex transmission is given by:

$$t_{max} = t_{Amax} + t_{SAmax} + t_{PAmax} + t_{WAmax} + t_{Bmax} + t_{SB}$$

where:

t_{Amax} = worst case time for device "A" to recognize that its receive buffer threshold has been crossed;

t_{SAmax} = maximum time for device "A" to transmit a maximum length MMC sysex (53 bytes). Since device "A" is not necessarily transmitting at full MIDI bandwidth, this time will be greater than or equal to 53 MIDI byte times (one MIDI byte time = .320msec).

t_{PAmax} = maximum time taken by device "A" to prepare the WAIT message after conclusion of its sysex transmission (quite possibly zero).

t_{WAmax} = maximum time for device "A" to transmit a WAIT (once again, greater than or equal to $6 \times .320$ msec).

t_{Bmax} = 10msec. We thus establish a fairly generous requirement that an MMC device must recognize the receipt of a WAIT message within 10msec.

t_{SB} = 16.96msec, since "B" is transmitting at full bandwidth, with each MIDI byte taking .320msec, and the maximum MMC sysex length is 53 bytes (48 data).

Since device "A" is receiving full bandwidth MIDI data for the duration of $\{t_{max}\}$, it follows that we will need at least $\{t_{max}/.320\}$ bytes of available space in "A"'s receive buffer in addition to the $\{H_A\}$ bytes below the "predetermined threshold". The actual value of $\{H_A\}$ need not be specified, but should be large enough so that reception of a single MMC sysex will not, in the majority of cases, cause a WAIT message to be generated.

One final element, $\{X_A\}$, represents the number of bytes that routines within device "A" are able to extract from the receive buffer for further processing during time $\{t_{max}\}$.

The minimum size in bytes, $\{Z\}$, of a device's receive buffer is therefore given by:

$$Z = H_A + \frac{t_{Amax} + t_{SAmax} + t_{PAmax} + t_{WAmax} + 26.96}{.320} - X_A$$

For example, if:

$$H_A = 53$$

$$t_{Amax} = 10\text{msec}$$

$$t_{SAmax} = 53 \times (.320 + .100) = 22.26 \text{ (i.e. max delay between MIDI byte transmissions is 100usec.)}$$

$$t_{PAmax} = 0$$

$$t_{WAmax} = 6 \times (.320 + .100) = 2.52$$

$$X_A = 0 \text{ (We temporarily ignore } X_A \text{ due to the difficulty of making reliable projections as to its value.)}$$

then

$$Z = 53 + \frac{10 + 22.26 + 0 + 2.52 + 26.96}{.320} - 0 = 245.9$$

In other words, given the conditions described, basic point to point MMC communications will operate quite satisfactorily with a receive buffer of 256 bytes.

External MIDI Mergers:

The typical use for a merger in an MMC system will be to collect responses from multiple Controlled Devices and feed them back to a Controller's MIDI In. This connection will have an impact on both Controller and Controlled Device:

At the Controller:

1. Clearly, for this type of merged connection to work at all, the Controller must not request data from the attached Controlled Devices in such a way that the maximum bandwidth of its single MIDI In port will be exceeded.
2. At the time that a Controller issues a WAIT command to the attached Controlled Devices, the merger may have already buffered up an arbitrary number of bytes which must yet be transmitted back to the Controller. In addition, under worst case conditions, each Controlled Device may have begun the transmission of a new sysex, and will not react to the WAIT until after transmission of the EOX. It is therefore difficult to establish a minimum buffer size for which a "buffer overflow" condition will never arise. Receive buffer capacities in the order of 1024 bytes, however, are expected to be adequate for Controllers supporting up to five attached devices through a single MIDI In port.

At the Controlled Device:

In a similar fashion, a WAIT message transmitted by a Controlled Device may be held up at the merger because sysex's from other devices are already pending transmission back to the Controller. Increasing the receive buffer size in a Controlled Device to 512 bytes is expected to serve adequately in systems which externally merge data from up to five Controlled Devices.

GENERAL MIDI SYSTEM LEVEL 1 DEVELOPER GUIDELINES

For Manufacturers and Composers
Second Revision, July 1998

Published by:
MIDI Manufacturers Association
Los Angeles CA

Prepared by Paul D. Lehrman, associate director for development
Center for Recording Arts, Technology & Industry, University of Massachusetts Lowell.

Additional text and editing by Howard Massey, MMA technical editor and
senior consultant, On The Right Wavelength

Copyright ©1996, 1998 MIDI Manufacturers Association

All rights reserved. No part of this document may be reproduced or transmitted in any form or by
any means, electronic or mechanical, including information storage and retrieval systems, without
permission in writing from the MIDI Manufacturers Association.

Second Printing 1998
Reformatted 2014

MMA
POB 3173
La Habra CA 90632

CONTENTS

Introduction.....	1
How to Use This Document.....	1
Acknowledgments	2
Changes to Specification in Second Revision	2
Additional Protocol Implementation Recommendations	3
Clarifications	3
Response to “GM System On” Message	4
Response to “Reset All Controllers” Message	5
RPN/NRPN Null Function Value	6
Use of Data Entry Controllers.....	7
GM Polyphony Requirements.....	7
GM Voice Allocation - Overflow and Channel Priority	8
Volume, Expression & Master Volume Response	9
Response to Pan	10
Use of Bank Select Messages	11
Response to Program Changes	12
Aftertouch	12
Built-In Effects & Response to Effects Controllers	13
Additional Notes About Controllers	14
Additional Instrument Sounds (Extensions to GM).....	14
Additional Drum Sounds & Kits (Extensions to GM)	15
Response to Note-off on Channel 10 (Percussion)	15
Mutually-Exclusive Percussion	16
File Formats and Editing Capability	17
MIDI Player Control: Starting in the middle	17
File Data: Prep bars	18
File Data: Pickup bars	18
File Data: SMF Marker Event.....	19
File Data: Other Meta-Events	19
File Data : Channel Assignments	19
File Data: Multiple Devices (non-GM hardware)	20

G M L e v e l 1 D e v e l o p e r G u i d e l i n e s - S e c o n d R e v i s i o n

Appendix A:	22
Voice editing	22
Appendix B:	23
Fat Labs Instrument Testing Specifications	23
General	23
Individual Timbres	23
Percussion	24

Introduction

This document was commissioned by the MIDI Manufacturers Association in 1995 to help developers of General MIDI System Level 1 (GM Level 1) products determine how to make their products compatible with as many other GM products as possible. Based on a survey of existing products, this document provides insights into areas of compatibility which are not clearly defined by the text of the formal GM Level 1 Specification (MMA0007/RP003), and adds additional recommendations to that specification based on market realities. This document should be used by manufacturers of GM compatible musical instruments (typically called "synthesizers") as well as by composers (authors of MIDI files) and developers of applications software to achieve more predictable playback of MIDI files.

Rather than attempt to redefine General MIDI "after-the-fact" (at this writing there are already over 1 million sound generating devices on the market which are designed for GM playback), this document identifies common practices and makes specific recommendations, yet still allows for freedom of creativity by individual manufacturers and developers.

The MMA GM Survey on which this document is based was commissioned to determine the current state-of-the-art in GM and to provide the information from which the MMA Technical Board could prepare recommendations. The work of collecting, compiling, and analyzing the survey data and making initial recommendations was done by Professor Paul Lehrman, a noted MIDI composer and author of numerous books and magazine articles on MIDI. Final editing and additional input was provided by Howard Massey, an industry MIDI consultant and author/educator. All recommendations were reviewed and evaluated by the MMA Technical Standards Board, to produce the final document you see here.

The survey results were published as part of the first printing of this document, but removed from the second (and on-line) publication(s).

How to Use This Document

This document begins with a summary of clarifications to the General MIDI System Level 1 specification. The summary clarifications are intended to be used as a companion and supplement to the actual specification. In many cases, the clarifications are additional information which the MMA Technical Board has determined should have been included in the published specification, but were omitted due to some oversight. In other cases this information is a clarification of ambiguous wording in the specification, and it is hoped that the new detail provided in this supplement will avoid further deterioration of GM compatibility.

The remainder of the document is an analysis of each issue of potential incompatibility (as determined by research into existing products). The issues are divided into two groups: "GM Synthesizer" issues and "GM Music File" issues. Within the GM Synthesizer group are any issues about the design of a GM compatible synthesizer that may be unclear, for both the manufacturer and the potential application or data developer. In the interest of making this a "quick reference guide," the analysis of each issue is preceded by a summary of the final recommendations, with separate recommendations for manufacturers of these products as well as for application or data developers where appropriate.

Appendix A lists common controls for voice editing, while Appendix B describes the procedure used by Fat Labs, a well known producer of music for computer games, to evaluate GM hardware for compatibility with music composed on Roland's Sound Canvas.

While the Sound Canvas is not an officially recognized reference for GM by the MMA, it is the predominant reference platform used by composers working in the field of interactive multimedia and PC games, as reported by the Interactive Audio Special Interest Group (IASIG). The appearance of the Fat Labs test procedure is not an endorsement of the test (nor the Sound Canvas) and is provided herein merely as additional information which developers may wish to use as a reference for determining an acceptable process for evaluating GM compatibility.

Acknowledgments

We would like to thank all of the MMA members who helped put the survey together; all who helped get others to respond; and all of those who responded. Special thanks to Tom Rettig, Yoshi Sawada, Mike D'Amore and Mike Kent. Extra special thanks to Barbara Blezard, administrative coordinator of the Recording Industry Environmental Task Force, based at the University of Massachusetts Lowell, who designed the database and entered the survey response data.

And thanks go to MMA Technical Board members Bob Lee (Gulbransen) and Rick Cohen (Kurzweil) for their efforts to edit and correct the recommendations and explanations herein.

Changes to Specification in Second Revision

The second revision documents a change in the MMA recommendation for correct response to the Reset All Controllers (CC #121) message. Expression was added to the list of controls which should be reset. The descriptions and/or placement in the document of some items to be reset or not reset were also rewritten for more clarity.

An additional paragraph was added to clarify the proper use of the Bank Select message in those files which are intended to support both General MIDI and extensions such as Roland's GS or Yamaha's XG devices.

Re: General MIDI System — Level 1 Specification

Additional Protocol Implementation Recommendations

- Data Entry Controllers (CC#6, CC#38)
- RNP/NRPN Null Function (C/A JMSC-0011)
- Mutually Exclusive use of Hi-hat and Triangle in Percussion Channel
- Response to Note-Off for Long Whistle and Long Guiro in Percussion Channel
- Response to All Sounds Off Message (cc120)

Clarifications

- Defined response to Turn GM System On Message
- Defined response to Reset All Controllers Message (*revised*)
- Defined Channel Priority scheme for Voice Allocation
- Defined Volume and Expression Controller response curves
- Defined Response to Pan Controller Messages
- Defined Response for Non-GM Controllers (Bank Select, Effects, etc.)
- Defined Response for Aftertouch (MIDI 1.0 Detailed Specification Recommendation)

(See text for specific details about each of these recommendations.)

Re: GM Synthesizer Issues

Response to “GM System On” Message

- The time required for complete response to the Turn GM System On message should be as short as possible (the current hardware average is 100ms).
- The response to this message should include the following actions on all Channels:
 1. All actions defined for the Reset All Controllers message (see next section) plus:
 - Set Volume (#7) to 100
 - Set Expression (#11) to 127
 - Set Pan (#10) to 64
 2. Devices which also respond to Effects Controllers (#91-#95) should reset to default values (power-up state). Effects are not required for GM but effects controllers may still be used by composers (see that section)
 3. Any other actions needed to restore the device to GM-compatible settings, such as reset Bank Select and Program Change to “0”.
- GM devices which support other modes of operation should “wake up” (power on) in GM mode, and not go out of GM Mode when receiving non-GM compatible messages (such as Bank Select).

File Player Recommendations: Devices designed only to play GM music files, or when in a mode which is designed only to play GM music files, should transmit the GM Mode On message upon power-up. Developers of software applications designed to play GM music files exclusively should ensure that the Turn GM System On message is transmitted when the application is launched, such as from a dialog prompting the user to connect and turn on the receiving device.

Composer Recommendations: Composers should not include the GM System On message in the body of GM music files. However, if necessary, the message can be included in “prep” bars (see page 21), as long as a delay of 100 - 200 ms before the onset of music is also provided.

Details:

Description of Issue:

After receiving a “GM System On” message, some devices need a period of time to reset themselves before they can start producing sound. How prevalent is this practice, what length pauses are required, and what can composers and authors do to avoid problems? In addition, some devices will go in or out of GM mode under certain conditions. What are these conditions, and are any of them acceptable?

Findings:

5 of the hardware respondents said that their devices don’t need any amount of time after receiving a Turn GM System On message before they can play sound. 2 were vague about the length of the pause needed, and of the others the range was from 10 ms all the way to “1-2 seconds”, with 100 ms being the median value.

Re: GM Synthesizer Issues

The most common time for a module to go into GM mode is when the unit powers up, which is as it should be. 11 devices go into GM mode the first time they're turned on, and 9 of those go into GM mode every time they're turned on. In addition, 7 of these can be put into GM mode from the front panel. 3 devices never go out of GM mode. 7 devices go out of GM mode when they receive specific Bank Select commands, but surprisingly, only 5 do so when they receive a Turn GM System Off message. One device goes out of GM mode when it receives a "GS Reset" (SysEx) command. 6 can go out of GM mode from the front panel.

Response to "Reset All Controllers" Message

- The MMA recommends that reception of the Reset All Controllers (CC #121) message cause the following response in GM devices:
 - Set Modulation (#1) to 0.
 - Set Expression (#11) to 127.
 - Set Pedals (#64-67) to 0.
 - Set Registered and Non-registered parameter LSB and MSB to null value (127).
 - Set Pitch Bender to center (64/0).
 - Reset Channel pressure to 0.
 - Reset polyphonic pressure to 0 (for all notes on that Channel).
 - All other controllers should be set to 0, otherwise the behavior should be documented.

The Association of Music Electronics Industry (AMEI, formerly the Japan MIDI Standard Committee) has proposed that the following parameters specifically be left unchanged upon receipt of a Reset All Controllers message. The MMA has not yet officially adopted this recommendation but it is unlikely that following this recommendation would cause serious incompatibilities. The MMA is expected to respond on this issue for both GM and non-GM devices shortly.

Do not Reset:

- Program change
 - Bank Select (#0 and #32)
 - Volume (#7)
 - Pan (#10)
 - Effects Controllers #91-95 (not a GM control)
 - Sound controllers #70-79 (not a GM Control)
 - Other Channel mode messages (#120, #122-#127)
-
- Manufacturers should create a section in their documentation for response to the Reset All Controllers message, especially for controllers used by the device on a global basis, if any.

Re: GM Synthesizer Issues

- If a device will respond differently to the Reset All Controllers message in General MIDI mode or in its native mode, this behavior should also be documented.

Details:

Description of Issue:

With the advent of many new controllers in MIDI, including Bank Select, the state of a controller after reset has become significant to compatibility issues as well as in performance situations. Exactly how should controllers in GM devices be reset following the reception of a Reset All Controllers (CC #121) or Turn GM System On SysEx message? This specific issue was not addressed in the MMA GM Survey, so this information has been obtained through an MMA proposal.

RPN/NRPN Null Function Value

- The RPN/NRPN Null Function (MIDI 1.0 Approved Protocol JMSC-0011) is not listed as a requirement in the GM Level 1 specification, but according to the MIDI protocol should be recognized by any MIDI device which also recognizes RPNs. Therefore this function is recommended for all GM Devices (and should be implemented in response to the Reset All Controllers message ... *see page 6*).

Composer Recommendations: See the MIDI Specification for instructions on when to use this message. Due to the apparent lack of implementation in current products, composers should not use this message if ignoring it will cause unacceptable playback.

Details:

Description of Issue:

RPNs (and NRPNs) are designed so that when a parameter is selected and then followed by a corresponding Data Entry (CC # 6/38) or Data Increment/Decrement (CC # 96/97) value, all subsequent data values will continue to address that parameter until another RPN or NRPN is selected. The “Null function” value (7F for both the MSB and LSB) is used to lock in the value of the RPN or NRPN so that subsequent Data Entry and Data Increment/Decrement commands are ignored. Some manufacturers, however, implement the Null function, or variations thereof, in response to other events or commands.

Findings:

Our survey determined that there is a confusion among manufacturers as to what exactly it means to “null” a Registered or Non-Registered Parameter. Apparently some thought the question applied to resetting values to their default settings (or zero). Respondents also indicated that the following commands/events affect the current RPN values, but did not always indicate which way. (Note: some devices respond to more than one of these, and some don’t respond to any of them):

Re: GM Synthesizer Issues

<u>Commands/Events that Affect RPN/NRPN Values (survey)</u>	<u>Respondents</u>
Power Off/On	3
GM Mode On or other mode change (CC#s 124-127)	3
Program Change	2
Reset All Controllers (CC#121)	2
RPN Null Function	2
Bank Select	1

The MMA does not yet have a recommendation on the appropriateness of response to these messages, other than Reset All Controllers (recommended).

Use of Data Entry Controllers

- Data Entry MSB (CC# 6) and LSB (CC #38) are required to be implemented by all GM devices for the adjustment of RPNs defined in the specification

Composer Recommendations: Composers adjusting RPNs should utilize the Data Entry MSB (CC# 6) and, if necessary, the Data Entry LSB (CC #38), not the Data Increment/Decrement (CC #96/97) for this purpose.

Details:

Description of issue:

According to the GM Level 1 Specification, a device must respond to RPNs 00-02. However, it is not specified how to *adjust* these parameters —by using Data Entry (CC #6 and CC #38), Increment/Decrement (CC #96/97), or by other means.

Findings:

9 of the 13 respondents indicated that their hardware devices support the Data Entry MSB (CC #6) for the adjustment of RPNs 00-02 (Pitch Bend Range, Fine Tuning, and Coarse Tuning, respectively). Only one respondent indicated that the Data Entry LSB (CC #38) was supported, and only one indicated support for the Reset RPN (7F/7F). Because the GM Level 1 Specification mandates response to RPNs for pitch bend sensitivity and tuning, support for the Data Entry MSB (CC# 6) and LSB (CC #38) is an oversight in the specification.

GM Polyphony Requirements

- GM devices should have as many voices (24 or more) as possible available at all times.
- GM Devices should diligently avoid “stacking” of voices (oscillators), so that composers can be sure that 24 note polyphony is available for all instrument sounds.
- Composers may want to limit their compositions to 16 simultaneous notes for compatibility with currently popular products which “stack” oscillators in contrast to the above recommendations.

Re: GM Synthesizer Issues

Details:

Description of Issue:

The GM Level 1 Recommended Practice specifies a minimum of either 24 “voices” for melodic and percussion sounds, or 16 for melody and 8 for percussion. Most people assume that this means that under all circumstances, a GM synthesizer that receives 24 MIDI note-on commands will produce 24 distinct sounds. However, a number of manufacturers, in an effort to improve sound quality without radically increasing the size of the sound engine, use two or more oscillators to create some programs. For example, Honky-tonk piano (program #4) is often created by layering two standard piano sounds and detuning them—a practice which results in total polyphony being reduced in half (from 24 to 12, for example). To compensate, manufacturers can raise the number of voices available, so that the chances of polyphonic overload are reduced. But even in those cases, unless the manufacturer provides at least 48 voices and no program uses more than two voices, polyphony of 24 cannot be guaranteed.

As far as composers are concerned, the question is, under current practice, how many voices are available in a General MIDI module to cover all possible scores? A secondary issue is whether it is desirable to separate percussion and non-percussion voices.

14 out of the 16 products described share polyphony between percussion and melody voices. Of the 2 others, one is a software-based synth engine (wavetables are loaded into a computer's RAM and played back, under MIDI control, directly through the CPU and a DAC), and one's answers were ambiguous. Of the 14 products that share polyphony, 10 provide 32 voices. Of the remainder, one provides 64 voices, but since it uses at least 2 voices on every program, the practical polyphony is reduced to 32.

The use of multiple voices to create GM programs is widespread. Only one respondent said that no programs used multiple voices — the others ranged from 15 to 55 (not including the “64-voice” system mentioned above), with the median value being around 30. The majority of systems have no programs that use 3 or more voices, and of those that do, the median value is 5.

GM Voice Allocation - Overflow and Channel Priority

- Allocation priority should always be given to the most recent voice(s) played. Second priority should be given to the loudest voice(s) currently being played. In addition, manufacturers should implement other musically-oriented solutions, such as stealing individual oscillators from multiple-oscillator programs, and reassigning oscillators which can no longer contribute substantially to the perception of a note.
- Notes on certain MIDI Channels should have priority over others. That is, Channel 10 receives highest priority, followed by Channel 1, then Channel 2, etc., with Channel 16 receiving lowest priority.

Re: GM Synthesizer Issues

Composer Recommendations: It is advised that MIDI file authors insure that voice overflow situations (where more than 24 notes need to be played simultaneously) are avoided. A MIDI file checker application can help identify these situations.

Details:

Description of issue:

Roland and Yamaha recommend specific Channel assignments for specific instrumentation, supposedly to provide better compatibility during a voice-overflow situation. If allocation schemes can contribute to compatibility problems, is there a recommendation for how notes (and Channels) should be prioritized?

Findings:

The respondents were split evenly between last-note and highest-volume priority. Three used various combinations of priorities. The issue of Channel priority was not addressed in the survey; however, many manufacturers including Roland and Yamaha follow the scheme described above, and this is also the scheme recommended by the IASIG for the upcoming Downloadable Sounds specification for synthesizers.

Volume, Expression & Master Volume Response

- Volume (CC#7) and Expression (CC #11) should be implemented as follows:

For situations in which only CC# 7 is used (CC#11 is assumed "127"):

$$L(\text{dB}) = 40 \log(V/127) \text{ where } V = \text{CC}\#7 \text{ value}$$

For example: CC#7 amplitude

127	0dB
96	- 4.8dB
64	-11.9dB
32	-23.9dB
16	-36.0dB
0	-*

This follows the standard "A" and "K" potentiometer tapers.

For situations in which both controllers are used:

$$L(\text{dB}) = 40 \log(V/127^2) \text{ where } V = (\text{volume} \times \text{expression})$$

The following table denotes the interaction of volume and expression in determining amplitude:

CC#7	CC#11	total amplitude	CC#7	CC#11	total amplitude
127	127	0dB	127	96	-4.8 dB
96	127	-4.8dB	127	64	-11.9dB
64	127	-11.9dB	127	32	-23.9dB
32	127	-23.9dB	127	0	-*
16	127	-36.0dB	64	64	-23.9dB
0	127	-*	32	96	-28.8dB

Re: GM Synthesizer Issues

- The Master Volume SysEx message is not specified in GM Level 1 (and the MMA is in the process of clarifying its application), so its implementation is optional at best.

Composer/Application Recommendations: Volume should be used to set the overall volume of the Channel prior to music data playback as well as for mixdown fader-style movements, while Expression should be used during music data playback to attenuate the programmed MIDI volume, thus creating diminuendos and crescendos. This enables a listener, after the fact, to adjust the relative mix of instruments (using MIDI volume) without destroying the dynamic expression of that instrument.

Details:

Description of Issue:

Use of volume (CC#7) and expression (CC#11) are required by GM Level 1, but there is a degree of confusion regarding their exact effect on sound levels, either singly or in combination. In addition, the Master Volume Universal System Exclusive command is not mentioned in GM Level 1, but is finding favor among some manufacturers.

Findings:

Not surprisingly, all hardware respondents said their devices responded to Continuous Controller #7. There was general agreement about how these two controllers interacted as well: In 9 cases, their values were combined (multiplied) to get the actual level. The recommended volume response curves for CC#7 (volume) and CC#11 (expression) used herein were provided to the General MIDI Working Group of the IASIG/MMA by Yamaha Corporation. Roland uses the same response curve, and other Japanese manufacturers who are members of the AMEI have agreed to do the same.

Of the 20 software respondents, all used CC#7 and 11 use CC#11. The largest number of software respondents (6) said they used CC#7 to set initial level and CC#11 for dynamic expression during music playback. In 2 other cases, CC#11 was set to a preset level and kept there.

Master Volume is supported by 6 hardware respondents, and 2 others have plans to include it in the future. Only 3 of the software respondents use it.

Response to Pan

- Immediate response to a Pan command (applying it to currently-sounding notes) should be supported.

Composer Recommendations: Until the above recommendation is universally implemented, composers of GM music files should be aware that sustained notes may not be panned on all devices.

Details:

Re: GM Synthesizer Issues

Description of issue:

Correct response to the Pan (CC #10) command is not defined in the GM specification document. Some devices respond by immediately shifting the apparent position of all currently sounding notes (on the Channel), while others will not move a current sound (choosing to Pan only those notes received after the Pan command).

Findings:

Software makers seem to be more optimistic than hardware would indicate. 12 of the 20 software respondents expect Pan commands to be acted upon immediately, while only 5 of the 14 hardware respondents said that sustained notes will be affected by a Pan command (although two more said they were planning to implement sustained-note Pan on their next models). Though one could imagine cases where not-panning would be nice, one should assume (by default) a timely response to a Pan message.

Use of Bank Select Messages

- Bank Select (CC #0/32) should be *completely ignored* in GM Mode.

Composer Recommendations: Composers of GM music files should not assume that any voices other than the GM Sound Set are available and should therefore not use Bank Select messages. Variations on GM voices can be accomplished by altering the playing style or by using controllers to introduce variations into the music

Details:

Description of issue:

GM Level 1 defines only a single Sound Set of 128 instruments and does not mandate the use of the Bank Select controller (CC #0/32). Yet many GM instruments provide additional sounds, accessed by Bank Select commands (sometimes followed by Program Changes), and some GM devices automatically go out of GM mode when Bank Select messages are received.

Findings:

12 of the 14 hardware respondents recognize the Bank Select MSB (CC #0). Of those, 4 also recognize the Bank Select LSB (CC #32). In 3 cases, they are followed by a Program Change command in order to call up a “variation” on the sound — this is the GS approach. In 7 cases they choose non-GM banks, and in at least one case they specifically take the device out of GM mode. These commands are much less frequently used on percussion Channels: only 2 of the devices recognize Bank Select on Channel 10, and then to select a non-drum bank for the Channel.

Re: GM Synthesizer Issues

Things are different on the software side: 12 of the software recipients do not use Bank Select at all. Of the others, 3 use CC#0, and 3 more also use CC#32.

Note: CC#0 and CC#32 were defined in 1990 as the Bank Select message and should not be used for any other purpose, separately or together. Transmission of only CC#0 or CC#32 is not a complete Bank Select message and should be discouraged.

Response to Program Changes

- Program Changes received during a sustained note should not cut off the note.

Composers/Application Recommendations: Despite the fact that most GM hardware will not cut off notes upon receipt of a Program Change message, the safest course of action is still to send these messages during silent portions of the music.

Details:

Some developers expressed concern that devices receiving a Program Change in the middle of a sustained note could cause the note to be cut off? This turned out to not be a problem—only 2 of the 14 hardware respondents report that a Program Change sent during a sustained note will cut off the note.

Aftertouch

- Channel Pressure (Aftertouch) response is a requirement of General MIDI, and should be used to add vibrato (or tremolo, if more appropriate) to voices. Manufacturers should assume that developers will use the *full* range of Aftertouch values, so high values of Aftertouch should not create unnatural amounts of vibrato.
- Other uses of Aftertouch such as volume or timbral change should be avoided.

Composer Recommendations: Composers of GM music files should not hesitate to use Aftertouch to add vibrato to voices; however, since there are no standards as to how much vibrato is to be applied with a given Aftertouch value, it is probably wise to err on the conservative side, lest listeners experience seasickness after a few bars. Some manufacturers also have chosen to implement Aftertouch as tremolo where appropriate to the instrument, so this should be considered by composers.

Details:

Description of Issue:

Response to Channel Pressure (Aftertouch) is a GM requirement, but no specifics are given as to how a device should respond.

Findings:

All but one manufacturer recognizes Aftertouch, and the majority of them use it to control either pitch-based vibrato depth or a more complex set of vibrato parameters.

Re: GM Synthesizer Issues

Five manufacturers said their Aftertouch response was programmable, and it is probably safe to assume that the default versions of their GM programs map Aftertouch to some form of vibrato.

Built-In Effects & Response to Effects Controllers

- Although not mandated by GM, manufacturers should feel free to provide onboard effects. The minimum suggested number of effects is two — Reverb and Chorus — though more may be provided, at the manufacturer's discretion.
- Controllers 91 and 93 should be used to set Reverb and Chorus send levels in order to maintain compatibility with current defacto standards (GS and XG, among others).
- Default effects send levels (those used on power-up or upon receipt of a GM Mode On message) should be moderate (value = 64 or less).
- Unassigned Controllers should not be used (to switch effects or for any other purpose).

Composer Recommendations: If including effects controllers can enhance the playback of a file, while at the same time the lack of effects will not harm it, then composers should feel free to use them. Because it cannot be assumed that effects send or return levels will default to any predictable value when a GM device is turned on or switched to GM mode, composers should place initial values — for safety purposes in the lower end of the range — for Controllers 91 and 93 on all Channels in “prep bars” at the beginning of music files.

Details:

Description of Issue:

Reverb, chorus, delay, flanging, EQ, etc. are to be found in just about every GM device on the market, since they can improve the sound significantly at relatively modest cost. GM Level 1, however, includes not a single mention of effects, and so manufacturers are on their own as to what effects to include, and how to make them accessible to the user.

Findings:

A majority of music industry respondents reported their products have effects. 8 of the respondents reported their devices could produce two effects simultaneously, and 3 said theirs could handle three or more. The GS (and base-level XG) usage of two effects — reverb and chorus — are most common in the GM community.

Virtually all of these devices set their reverb and chorus send levels via CC#91 and CC#93, respectively. Software respondents are a bit more conservative; only 7 use these controllers in their files. Though other effects are available in some of the hardware, none of them were addressed by the software respondents.

Only one hardware respondent reported that the effects on their devices are not adjustable. 3 said they were adjustable on a global or per-program basis, while 9 said

Re: GM Synthesizer Issues

they were adjustable on a per-Channel basis. Several hardware respondents use unassigned controllers or NRPN's to select effects programs or variations, while others use SysEx messages.

The default settings of the effects varied widely among the hardware respondents, and effects are not required for GM, so no obvious recommendation (other than to follow the Sound Canvas and XG guidelines) is evident from this survey.

Additional Notes About Controllers

- Factory presets should generally be set up with all controllers (except Volume [CC #7] and Expression [CC #11]) set to 0 or 64 (center), as the case may be.
- The All Sound Off (CC#120) Channel Mode message is recommended for all MIDI devices, though not listed as a requirement in the GM System Level 1 specification.
- Non-standardized adjustments should be made with NRPNs or SysEx, as defined by the MIDI Specification.

Composer Recommendations: Composers of GM music files should assume that the only controllers available to them are those listed in GM Level 1 — with the addition of Data Entry (CC#6 and, if necessary, CC #38) for adjusting RPNs. Use of non-standard controllers for special purposes should be restricted to systems (applications) where the MIDI data is not intended for playback on other systems.

Details:

Appendix A presents a detailed listing of the controllers used in surveyed GM hardware devices. What is clear from this data is that virtually all such devices support all the controllers and Channel Mode messages required by GM Level 1. Most also implement those described in Roland's GS and Yamaha's base-level XG command sets (including Bank Select [CC #0/32], Data Entry [CC #6/38], Sostenuto [CC #66], Soft Pedal [CC #67], Reverb Send Level [CC #91], and Chorus Send Level [CC #93]).

Among software respondents, 2 replied that they include RPNs in their files, and one replied that they use "unassigned" (non-defined) controllers for internal functions. Appendix B presents a summary of common NRPN messages for "voice editing".

Additional Instrument Sounds (Extensions to GM)

- Additions (or variations) to the GM Sound Set should not be accessible while the device is in GM mode.
- In order to maintain some degree of standardization among GM devices, organization of and access to variation sounds may be most appropriate if done in the manner of GS and/or XG specifications.

Composer Recommendations: Composers of GM music files should assume there are no other sounds available besides the GM Sound Set, unless they are writing for specific platforms.

Re: GM Synthesizer Issues

Details:

Exactly half of the devices surveyed contain from anywhere from 32 to 500+ additional sounds besides the GM sound set. There was no apparent consensus on how these sounds are laid out: only two manufacturers (besides Roland) mentioned GS's "variations" scheme, and the rest use their own structure. Yamaha has since introduced a largely compatible structure with their XG format, but there are sufficient variations between XG and GS (and GM) to require an additional document describing this issue.

Additional Drum Sounds & Kits (Extensions to GM)

- Extra drum sounds (additions or variations to the GM Percussion Map) should not be accessible while the device is in GM mode.
- In order to maintain some degree of standardization among GM devices, organization of and access to variation sounds may be most appropriate if done in the manner of GS and/or XG specifications.

Composer Recommendations: Composers of GM music files should not use variation kits (via program changes, etc.) or extended drum notes unless absolutely sure that they will not result in unacceptable degradation of the performance on dissimilar instruments. For example, the gentle Roland brush snare sound will be replaced with a strong snare hit on most GM devices, which would generally be unacceptable to the composer.

Details:

The GS approach to drum kits is very popular: 7 devices provide extra sets in conformity with the GS guidelines, and subsets or approximations of the GS sets are found in 8 more. Only 2 devices reported that they had no additional drum sounds besides the standard GM set. But how the additional sets are accessed is not as clear cut. 6 use Program Change commands (as per GS), 2 use Bank Select by itself (a practice specifically prohibited by the MIDI Specification), and 2 use a combination of the two. Almost all devices use notes outside of the GM Percussion Map range to access additional sounds, but it is unclear if there is any consensus therein.

Response to Note-off on Channel 10 (Percussion)

- Only those two GM percussion sounds whose duration is most naturally under player control — long whistle and long Guiro — should respond to note-offs on Channel 10.
Note: The MIDI Specification requires that all Note-On commands have a corresponding Note-Off command, and it is assumed that all MIDI transmitters will comply with this requirement)

Composer Recommendations: There is little harm (musically speaking) sending a note-off message to a drum (one-shot) that will be ignored, but composers of GM music files should also assume that these messages may not work on all GM devices and author accordingly.

Details:

Re: GM Synthesizer Issues

Description of Issue:

In musical context, percussion sounds are typically of defined length, in comparison to other instruments which have a variable sustain segment under composer control. Are any of the percussion sounds in a GM device cut off upon receipt of a Note-off command? This is of particular importance if composing percussion tracks with drum pads which have a short, fixed-length, note duration.

Findings:

Only 3 of the 14 respondents said that any of their GM percussion sounds respond to note-off commands. (Somewhat more said this applies to non-GM percussion sounds.) The sounds in question, when they were specified, included whistle, long Guiro, and open cymbals, with no apparent consistency.

Mutually-Exclusive Percussion

- Two mutually-exclusive groups for drum sounds are recommended: open/pedal/closed hi-hat and open/mute triangle. Additional groups of mutually-exclusive drum sounds may be included as long as those groupings make sense musically.

Composer Recommendations: Composers of GM music files can assume that the two above-named mutually-exclusive groups are supported by GM devices but should not assume the presence of other groups.

Details:

Description of Issue:

In order to support realism expectations, manufacturers set up certain groups of sounds in the percussion set to be mutually exclusive, so that playing a sound in the group cuts off any other previously-played sound in the group (as would naturally happen).

Findings:

12 of the 14 respondents use one or more mutually-exclusive groups for their GM percussion sounds. GS, for example, mandates several mutually exclusive groups: high/low whistle, long/short Guiro, “open/mute” cuica, open/mute triangle, and open/pedal/closed hi-hat. (Another pair, open/mute surdo, uses sounds not included in the GM Percussion Map.) Besides Roland, one other manufacturer includes all of the GS groups. 2 others include 3 of the GS groups, and 2 more include 2 of the GS groups (specifically hi-hat and triangle). 5 more have mutually-exclusive groups but did not specify what they were, and one allows users to define their own groups.

Re: MMA GM Music File Issues

File Formats and Editing Capability

- SMFs should be considered editable, and if composers want their files *not* to be editable by users, they should use proprietary formats.
- Files which may be edited should avoid Program Changes within the body of the music, which could be lost in editing, resulting in playback with incorrect sounds. Likewise, authors should avoid controllers or notes hanging over bar lines, which could result in unexpected articulation and stuck notes after editing.

The issue: What provisions need to be made so that GM files can be edited by users? If a GM music file is user-editable, how can you enable chunks of files to be moved around while making sure that Program Changes and controller messages are preserved in their correct places? Can editing be prevented?

Findings: Exactly half of the software respondents said their files were not editable by users. Of these, 4 distribute files in a proprietary (i.e., non-translatable) format, and 5 are read-only files, either on a CD-ROM, hidden in a CD-ROM or Windows Resource, or in an unspecified form. One respondent puts a copyright notice meta-event into their files.

Of the files that are editable, all are provided in SMF format. Only one respondent stated their files are in Type 1 format, and one stated their files are in Type 0 format; the rest did not specify a type.

Only one respondent said that documentation is included with their software explaining the issues for editing and providing instructions on how to deal with them. One respondent said “it should be up to the sequencer software”.

MIDI Player Control: Starting in the middle

- MIDI “player” or driver software should determine whether or not a GM music file can be started in the middle.
- Players should be capable of chasing controllers and program changes, either by scanning the file backwards from the starting point (“walking” the file) or by using special setup files. If using the former method, the scanning code must be fast and have a high level of priority in order to avoid long delays.

The issue: Since MIDI is a serial data stream, special care must be taken when starting a sequence in the middle, so that crucial commands that normally appear at the beginning of the sequence, such as Program Changes and controller settings, are not overlooked when the music starts to play.

Findings: 6 software application respondents had a simple answer to this: they don’t allow files to be started in the middle. Of those who do, 5 said before they play the file they scan (or “walk”) the file backwards from the designated starting point, and transmit appropriate commands as they are found. 3 allow users to start the files at specific markers, and when they do, a special short setup file containing the necessary information is transmitted first.

Re: MMA GM Music File Issues

File Data: Prep bars

- Prep bars should not be used where synchronization and exact starting time is an issue.
- Prep bars should be as short as possible (a few clock ticks should suffice in most situations)

The issue: It can be helpful for a GM sequence to be preceded with a preparatory (or “prep”) bar of some length so that the Turn GM System On message can be transmitted and initial values for controllers, Program Changes, pitchbend, Aftertouch, and other parameters can be set before the music starts to play. But prep bars can cause problems when there are timing issues to be considered, such as if MIDI file is used in sync with some visual. Should such a bar be used, and, if so, how long should it be?

Findings: There is a lot of variation found in how prep bars are used. Of those who use them, 6 use one complete 4/4 bar or more, while 4 try to minimize the amount of time necessary to start a file by making the prep bar very short, measured in a few clock “ticks” — for example, 9 ticks where 1 tick = 1/480 beat.

Obviously if music is to be synced with visuals or used for scene transitions — prep bars should not be used, or at least kept as short as possible. Composers should bear in mind that rarely will all 16 MIDI Channels be called upon to play on the downbeat of a sequence, so only the tracks playing at the beginning need to be initialized right away, and the time required to do that in most cases will be negligible. The initialization information for other tracks can be transmitted after the sequence starts, as long as it is sent before they need to play.

File Data: Pickup bars

- Pickup bars should be as brief as possible, set to the minimal time signature required (generally, 1/4 or 3/8 will suffice).
- A time signature meta-event should be inserted at the end of the pickup bar in order to set the correct time signature for the body of the music to follow.

The issue: Often a sequence will start out with a “pickup” — a group of notes shorter than a bar line that precede the first bar, such as the three eighth-notes at the beginning of “Seventy-Six Trombones.” Should this pickup be in a short bar by itself, or should it be the last part of a standard-length bar which has blank space at the beginning?

Findings: 5 software respondents said they use a full bar at the beginning of the sequence, and leave the beats before the pickup blank. 4 said they give the pickup bar its own time signature, equivalent to its length (in the above example, 3/8), and then change the time signature for subsequent bars. 4 said that it depended on the situation, and 3 on whether the start time or synchronization of the sequence was critical. One said they don’t concern themselves with the barlines at all, and just “let the notes fall where they may”.

Re: MMA GM Music File Issues

File Data: SMF Marker Event

- Markers (an SMF Meta Event) may be used for any purpose, but a specific response to markers should not be assumed.

The issue: The correct use of Markers (Meta-Events under the Standard MIDI Files Recommended Practice) is unclear in the SMF document. How are they being used?

Findings: 3 software respondents said they use markers along with a short setup file (as described in the previous topic) for alternate starting points. 2 use them to synchronize sound effects, visuals, etc. (audio and other non-MIDI events are being considered for inclusion in a future version of SMF). 2 use them for internal purposes, and 2 will use them in their products' "next evolution." One respondent uses them to designate loop points, and another puts them on a separate track where they denote program changes (no details were given on this rather odd statement).

File Data: Other Meta-Events

- Copyright information (text) should be placed in the MIDI file using the Copyright Meta-Event.
- Song Titles should be placed on the MIDI file using the Sequence/Track Name event on the first track or in a Type 0 file.
- Meta-Events should not be used for proprietary purposes (except when used in closed systems such as video game consoles where the files can not be played on an incompatible device.)

The issue: How are other Meta-Events being used?

Findings: The largest number of respondents by far — 7 — are using the Copyright Notice Meta-Event. One other is putting a copyright notice within the track name list. 5 are using Meta-Events for unspecified or internal purposes. 4 are using Lyrics. 3 are using Track Names, and 3 are using "Titles," presumably Sequence/Track Name on the first track or in a Type 0 file. One is using Cue Points for alternate starting points.

File Data: Channel Assignments

- No specific assignment scheme (including the many GS, XG and variants) can be recommended as a sole scheme for all musical performances. However, new Meta-Events could be added to the Standard MIDI File Recommended Practice in future to identify parts regardless of Channel, allowing file players to intelligently map parts to playback Channels as needed to provide the best possible user interaction.

The issue: Besides the restriction that Channel 10 be reserved for key-based percussion, are there any other ways to designate Channels for specific instruments in a General

Re: MMA GM Music File Issues

MIDI sequence that make sense? That way, a user (or a hardware file player) can know how tracks are assigned (and which to mute or solo, for example)?

Findings: 11 respondents said they use no special designations for MIDI Channels in a sequence. Of the others, each had their own idea of how Channels should be used.

Since each file is designed for a different purpose, there is no assignment method which would serve all users. Instead, a recent proposal is that the file could be encoded with information which indicates which of the common musical “parts” appears on which Channels at which times. For example, the file could include an event which reports that following segment on Channel 2 contains the Bass Part, while the segment on Channel 3 is the Right-Hand Piano Part, Channel 4 is the Left-Hand Piano, etc. This can be changed throughout the piece as necessary, and the parts may be assigned to totally different Channels in another piece. This will enable devices (and users) to easily determine what parts to play/mute (etc.) in music-minus-one, music education, and other applications. However, this is not yet an MMA recommended practice.

File Data: Multiple Devices (non-GM hardware)

- Authors wishing to address multiple platforms should create different files for the different platforms.
- Files authored to Microsoft’s (now defunct) “Dual-Mode” format should be marked using Microsoft’s Mark-MIDI utility.
- Files authored for Roland GS or Yamaha XG should include the appropriate (GS or XG) reset events, but the GM System On message is still required for all GM devices. See Roland or Yamaha guidelines for the correct usage of these messages.
- MIDI file players should be capable of remapping music files based upon the identity and configuration of the target device (new Meta-Events may be added to the Standard MIDI File Recommended Practice for this purpose).

The issue: Composers writing for PC applications may need to write music that can be played on multiple (not just GM) formats. Should they combine all of the formats they want to address in one file.

Findings: Apparently there are many ways to address this issue. Three respondents produce different versions for the various platforms they want to address. Three depend on system software (i.e. Microsoft’s “MIDI Mapper”) to send data to different devices. Three respondents use proprietary Channel-mapping schemes and/or voice-allocation algorithms. One respondent uses a single format and depends on the software driver to mimic General MIDI on other formats. Yet another respondent claims that tracks for different platforms are assignable by a port event.

Microsoft’s Multimedia PC (MPC) specification for Windows 3.0 and 3.1 called for the use of a “Mark-MIDI” flag to specify if the file included “Base” (Channels 13-16, with percussion on 16) or “Extended” (Channels 1-10, with percussion on 10) performance data (or both). The Windows 95 multimedia documentation, however, recommends that this

Re: MMA GM Music File Issues

flag be omitted from music files, since GM Level 1 (using all 16 Channels, with percussion on 10) is specified as being the minimum performance standard.

Microsoft's MIDI File Player (MCI Sequencer) has been written to recognize MPC formatted files and ignore the base data in favor of playing only the extended data on the GM device (eliminating the need to disable certain Channels to avoid doubling-up of instruments). This practice should be followed by all file players, and be extended to apply to GS and XG file formats as well.

APPENDIX A:

Voice editing

The important issue of GM voice and drum sound editing may become standardized in the future, at which time RPNs will likely be assigned for that purpose. Until that time, manufacturers are free to assign NRPNs (the “proprietary” equivalent of RPNs) for this purpose. The following NRPNs are common to both GS and XG, but will also likely have an unpredictable effect, or no effect, on GM products:

MSB	LSB	Description
01h	08h	Vibrato rate
01h	09h	Vibrato depth
01h	0Ah	Vibrato delay
01h	20h	Filter cutoff frequency
01h	21h	Filter resonance
01h	63h	Envelope attack rate
01h	64h	Envelope decay rate
01h	66h	Envelope release rate
18h	rr	Pitch coarse of specified drum sound
1Ah	rr	Level of specified drum sound
1Ch	rr	Panpot of specified drum sound
1Dh	rr	Reverb send level of specified drum sound
1Eh	rr	Chorus send level of specified drum sound

Note: rr = drum sound note number

APPENDIX B:

Fat Labs Instrument Testing Specifications

The following outlines the testing requirements for Fat Labs Certification. Fat Labs certifies sound cards to be compatible with music written for GM instruments (specifically the Roland Sound Canvas). Their testing process is provided here as a reference for companies curious about what has already been done to validate GM compatibility, but is not an endorsement by the MMA of this particular process.

General

- a. The instrument system must comply with the Level 1 General MIDI spec.
- b. Polyphony lower than 24 simultaneous voices will be acceptable if listening to test files reveals that the formula used for dynamic voice allocation gives a suitable performance.
- c. The instrument system must default at power-up to General MIDI mode.
- d. The instrument system must default at power-up to a bend range equal to 2 half-steps.
- e. The instrument system must respond to controllers for Mod Wheel, Volume, Pan, Sustain Pedal, Pitch Bend Range, and All Notes Off. If it is claimed that the instrument system has reverb and chorus, it must respond to controllers for those effects as well.
- f. The instrument system must respond to Controller 7 while notes are sustained.
- g. For uses in advanced DOS games, the instrument system must have an MPU-401 interface in hardware, or must place minimal enough demands on the host system that we can be reasonably sure that all known software using General MIDI will work with the instrument system.

Individual Timbres

After the instrument system is set to a reference level and tuning standard, each timbre will be subjected to the following tests:

- a. Volume at Velocity 64, Different Octaves. For each octave (4 are tested: middle C, 2 octaves below, 2 octaves above, and one note that floats to 4 octaves above or below middle C, depending on the expected range of the instrument), a velocity offset of not more than ± 5 must produce a perceived volume that matches the Sound Canvas for most listeners.
- b. Volume at C3, Different Velocities. For each velocity (3 are tested: 17, 64, and 127), a velocity offset of not more than ± 5 must produce a perceived volume that matches the Sound Canvas for most listeners.
- c. Envelope. Time values for A, D, and R must be within 10% of the Sound Canvas's; for sustain, a velocity offset of not more than ± 5 must produce a perceived volume that matches the Sound Canvas for most listeners.
- d. Intonation. Intonation should be within ± 5 cents of the Sound Canvas. For timbres such as "Honky-Tonk Piano," that vary from the reference pitch, the instrument may be less detuned than the Sound Canvas.

Percussion

Each percussion instrument will be subjected to the following tests:

- a. Volume at C3, Different Velocities. For each velocity (3 are tested: 17, 64, and 127), a velocity offset of not more than ± 5 must produce a perceived volume that matches the Sound Canvas for most listeners.
- b. Envelope. Time values for A, D, and R must be within 10% of the Sound Canvas's; for sustain, a velocity offset of not more than ± 5 must produce a perceived volume that matches the Sound Canvas for most listeners.
- c. Panning. Percussion instruments must be panned to produce a perceived placement that, for most listeners, matches that of the Sound Canvas.

APPENDIX

MIDI 1.0 Detailed Specification Addenda [post 1996] as of December 2013

- The following changes/additions were approved by MMA/AMEI since the "96.1 Second Edition" publication:
 - [Response to Reset All Controllers](#)
 - [Response to Data Increment/Decrement Controller](#)
 - [Sound Controller Defaults \(Revised\)](#)
 - [Renaming of CC91 and CC93](#)
 - [File Reference SysEx Message \(.pdf\)](#)
 - [Sample Dump Size/Rate/Name Extensions \(.pdf\)](#)
 - [Controller Destination SysEx Message \(.pdf\)](#)
 - [Key-based Instrument Controller SysEx Message \(.pdf\)](#)
 - [Global Parameter Control SysEx Message \(.pdf\)](#)
 - [Master Fine/Course Tuning SysEx Message \(.pdf\)](#)
 - [Redefinition of RPN01 and RPN02 \(Channel Fine/Course Tuning\)](#)
 - [RPN05 Modulation Depth Range \(.pdf\)](#)
 - [Extension 00-01 to File Ref SysEx Message \(.pdf\)](#)
 - [Default Pan Formula](#)
 - [High Resolution Velocity Prefix \(.pdf\)](#)
 - [Three Dimensional Sound Controllers \(.pdf\)](#)
- These changes/additions were made to the [SMF Specification](#) since the "96.1 Second Edition" publication:
 - [SMF Lyric Events Definition](#)
 - [SMF Device/Program Name Meta-events](#)
 - [SMF Language and Display Extensions](#)
 - [XMF Patch Prefix Meta Event](#)
- These changes/additions were made to the [MIDI Tuning Specification](#) since the "96.1 Second Edition" publication:
 - [Scale/Octave Tuning w/Defaults](#)
 - [MIDI Tuning Bank/Dump Extensions](#)
- The [General MIDI Specification](#) was updated to Level 2, and a General MIDI "Lite" version (intended for cell phones) was also developed. See the GM section of [www.midi.org](#) for more details.

-
- **Note:** In addition to the above, MMA has since published the following additional Specifications and Recommended Practice documents which are available separately:
 - [Downloadable Sounds \(DLS\)](#) (including DLS, DLS2, and Mobile DLS)
 - [MIDI for IEEE-1394 \(pdf\)](#)
 - [eXtensible Music Format \(XMF\)](#) (including XMF, mobile XMF, and related RPs)
 - [Scalable Polyphony MIDI \(SP-MIDI\)](#)
 - [MIDI XML Names DTDs](#)
 - [Mobile Phone Control \(pdf\)](#)
 - [Mobile Musical Instrument \(pdf\)](#)
 - [MIDI Visual Control \(pdf\)](#)

Please visit our [Specifications](#) page on [www.midi.org](#) for the latest information and links to all current MMA Specifications and Recommended Practices.