The Complete MIDI 1.0 Detailed Specification

Incorporating all Recommended Practices

document version 96.1
third edition

IMPORTANT NOTE

This publication represents the complete documentation of
The MIDI Specification and all related Recommended
Practices as of 1996. For subsequent corrections and
additions visit https://www.midi.org/specifications.

The original "version 96.1" document produced in 1996 was a combination of the latest revisions of the various MIDI specification documents authored and published by the MMA 1985-1996. The Second Edition (1998) added the GM Developer Guidelines (also available on the MMA web site). A correction was made to the Tutorial in 2006 (without changing the Edition number) and the GM Developer Guidelines were reformatted in 2014 when preparing this PDF document (Third Edition) for publication. This Edition also includes a reference list of Changes and Addenda made between 1996 and the date of publication (see Appendix).

SECTION CONTENTS

Note: This publication represents the complete documentation for The MIDI Specification and all related Recommended Practices as of 1996. Please visit www.midi.org/specifications for subsequent corrections and additions.

Tutorial on MIDI
and Music Synthesis

Rev. April 2006: Corrected NRPN/RPN text on Page 6.

Tutorial on MIDI and Music Synthesis
Written by Jim Heckroth, Crystal Semiconductor Corp.
Used with Permission

Windows is a trademark of Microsoft Corporation. MPU-401, MT-32, LAPC-1 and Sound Canvas are trademarks of Roland Corporation. Sound Blaster is a trademark of Creative Labs, Inc. All other brand or product names mentioned are trademarks or registered trademarks of their respective holders.

Tutorial on MIDI and Music Synthesis

The Musical Instrument Digital Interface (MIDI) protocol has been widely accepted and utilized by musicians and composers since its conception in the 1982/1983 time frame. MIDI data is a very efficient method of representing musical performance information, and this makes MIDI an attractive protocol not only for composers or performers, but also for computer applications which produce sound, such as multimedia presentations or computer games. However, the lack of standardization of synthesizer capabilities hindered applications developers and presented new MIDI users with a rather steep learning curve to overcome.

Fortunately, thanks to the publication of the General MIDI System specification, wide acceptance of the most common PC/MIDI interfaces, support for MIDI in Microsoft WINDOWS and other operating systems, and the evolution of low-cost music synthesizers, the MIDI protocol is now seeing widespread use in a growing number of applications. This document is an overview of the standards, practices and terminology associated with the generation of sound using the MIDI protocol.

MIDI vs. Digitized Audio

Originally developed to allow musicians to connect synthesizers together, the MIDI protocol is now finding widespread use as a delivery medium to replace or supplement digitized audio in games and multimedia applications. There are several advantages to generating sound with a MIDI synthesizer rather than using sampled audio from disk or CD-ROM. The first advantage is storage space. Data files used to store digitally sampled audio in PCM format (such as .WAV files) tend to be quite large. This is especially true for lengthy musical pieces captured in stereo using high sampling rates.

MIDI data files, on the other hand, are extremely small when compared with sampled audio files. For instance, files containing high quality stereo sampled audio require about 10 Mbytes of data per minute of sound, while a typical MIDI sequence might consume less than 10 Kbytes of data per minute of sound. This is because the MIDI file does not contain the sampled audio data, it contains only the instructions needed by a synthesizer to play the sounds. These instructions are in the form of MIDI messages, which instruct the synthesizer which sounds to use, which notes to play, and how loud to play each note. The actual sounds are then generated by the synthesizer.

For computers, the smaller file size also means that less of the PCs bandwidth is utilized in spooling this data out to the peripheral which is generating sound. Other advantages of utilizing MIDI to generate sounds include the ability to easily edit the music, and the ability to change the playback speed and the pitch or key of the sounds independently. This last point is particularly important in synthesis applications such as karaoke equipment, where the musical key and tempo of a song may be selected by the user.

MIDI Basics

The Musical Instrument Digital Interface (MIDI) protocol provides a standardized and efficient means of conveying musical performance information as electronic data. MIDI information is transmitted in "MIDI messages", which can be thought of as instructions which tell a music synthesizer how to play a piece of music. The synthesizer receiving the MIDI data must generate the actual sounds. The MIDI 1.0 Detailed Specification provides a complete description of the MIDI protocol.

The MIDI data stream is a unidirectional asynchronous bit stream at 31.25 Kbits/sec. with 10 bits transmitted per byte (a start bit, 8 data bits, and one stop bit). The MIDI interface on a

MIDI instrument will generally include three different MIDI connectors, labeled IN, OUT, and THRU. The MIDI data stream is usually originated by a MIDI controller, such as a musical instrument keyboard, or by a MIDI sequencer. A MIDI controller is a device which is played as an instrument, and it translates the performance into a MIDI data stream in real time (as it is played). A MIDI sequencer is a device which allows MIDI data sequences to be captured, stored, edited, combined, and replayed. The MIDI data output from a MIDI controller or sequencer is transmitted via the devices' MIDI OUT connector.

The recipient of this MIDI data stream is commonly a MIDI sound generator or sound module, which will receive MIDI messages at its MIDI IN connector, and respond to these messages by playing sounds. Figure 1 shows a simple MIDI system, consisting of a MIDI keyboard controller and a MIDI sound module. Note that many MIDI keyboard instruments include both the keyboard controller and the MIDI sound module functions within the same unit. In these units, there is an internal link between the keyboard and the sound module which may be enabled or disabled by setting the "local control" function of the instrument to ON or OFF respectively.

The single physical MIDI Channel is divided into 16 logical channels by the inclusion of a 4 bit Channel number within many of the MIDI messages. A musical instrument keyboard can generally be set to transmit on any one of the sixteen MIDI channels. A MIDI sound source, or sound module, can be set to receive on specific MIDI Channel(s). In the system depicted in Figure 1, the sound module would have to be set to receive the Channel which the keyboard controller is transmitting on in order to play sounds.
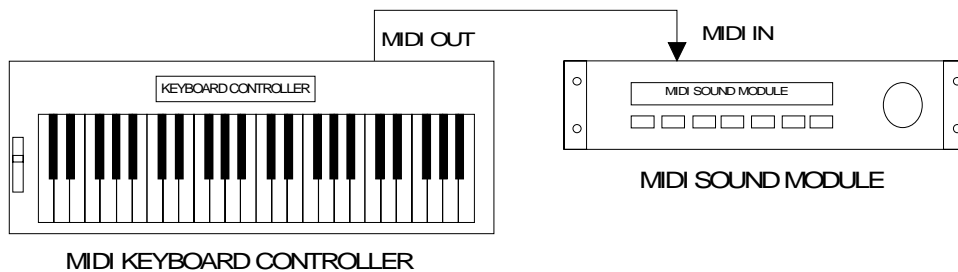
Figure 1. A Simple MIDI System

Information received on the MIDI IN connector of a MIDI device is transmitted back out (repeated) at the devices' MIDI THRU connector. Several MIDI sound modules can be daisy-chained by connecting the THRU output of one device to the IN connector of the next device downstream in the chain.

Figure 2 shows a more elaborate MIDI system. In this case, a MIDI keyboard controller is used as an input device to a MIDI sequencer, and there are several sound modules connected to the sequencer's MIDI OUT port. A composer might utilize a system like this to write a piece of music consisting of several different parts, where each part is written for a different instrument. The composer would play the individual parts on the keyboard one at a time, and these individual parts would be captured by the sequencer. The sequencer would then play the parts back together through the sound modules. Each part would be played on a different MIDI Channel, and the sound modules would be set to receive different channels. For example, Sound module number 1 might be set to play the part received on Channel 1 using a piano sound, while module 2 plays the information received on Channel 5 using an acoustic bass sound, and the drum machine plays the percussion part received on MIDI Channel 10.

MIDI and Wavetable Synthesis Tutorial

Figure 2. An Expanded MIDI System

In this example, a different sound module is used to play each part. However, sound modules which are "multitimbral" are capable of playing several different parts simultaneously. A sin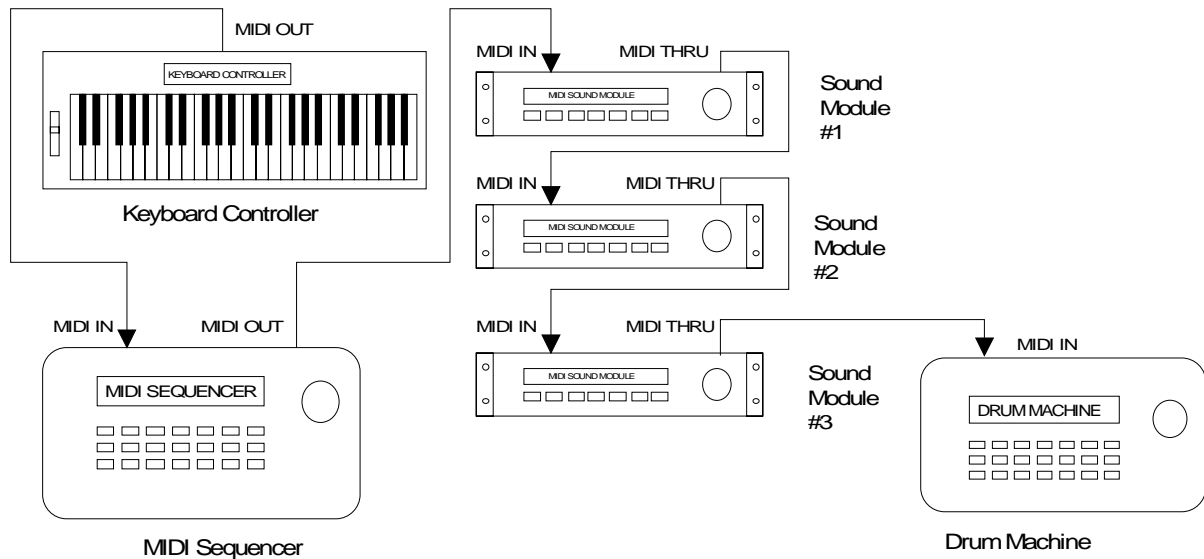gle multitimbral sound module might be configured to receive the piano part on Channel 1, the bass part on Channel 5, and the drum part on Channel 10, and would play all three parts simultaneously.

Figure 3 depicts a PC-based MIDI system. In this system, the PC is equipped with an internal MIDI interface card which sends MIDI data to an external multitimbral MIDI synthesizer module. Application software, such as Multimedia presentation packages, educational software, or games, sends MIDI data to the MIDI interface card in parallel form over the PC bus. The MIDI interface converts this information into serial MIDI data which is sent to the sound module. Since this is a multitimbral module, it can play many different musical parts, such as piano, bass and drums, at the same time. Sophisticated MIDI sequencer software packages are also available for the PC. With this software running on the PC, a user could connect a MIDI keyboard controller to the MIDI IN port of the MIDI interface card, and have the same music composition capabilities discussed in the last two paragraphs.

There are a number of different configurations of PC-based MIDI systems possible. For instance, the MIDI interface and the MIDI sound module might be combined on the PC add-in card. In fact, the Multimedia PC (MPC) Specification requires that all MPC systems include a music synthesizer, and the synthesizer is normally included on the audio adapter card (the "sound card") along with the MIDI interface function. Until recently, most PC sound cards included FM synthesizers with limited capabilities and marginal sound quality. With these systems, an external wavetable synthesizer module might be added to get better sound quality. Recently, more advanced sound cards have been appearing which include high quality wavetable music synthesizers on-board, or as a daughter-card options. With the increasing use of the MIDI protocol in PC applications, this trend is sure to continue.
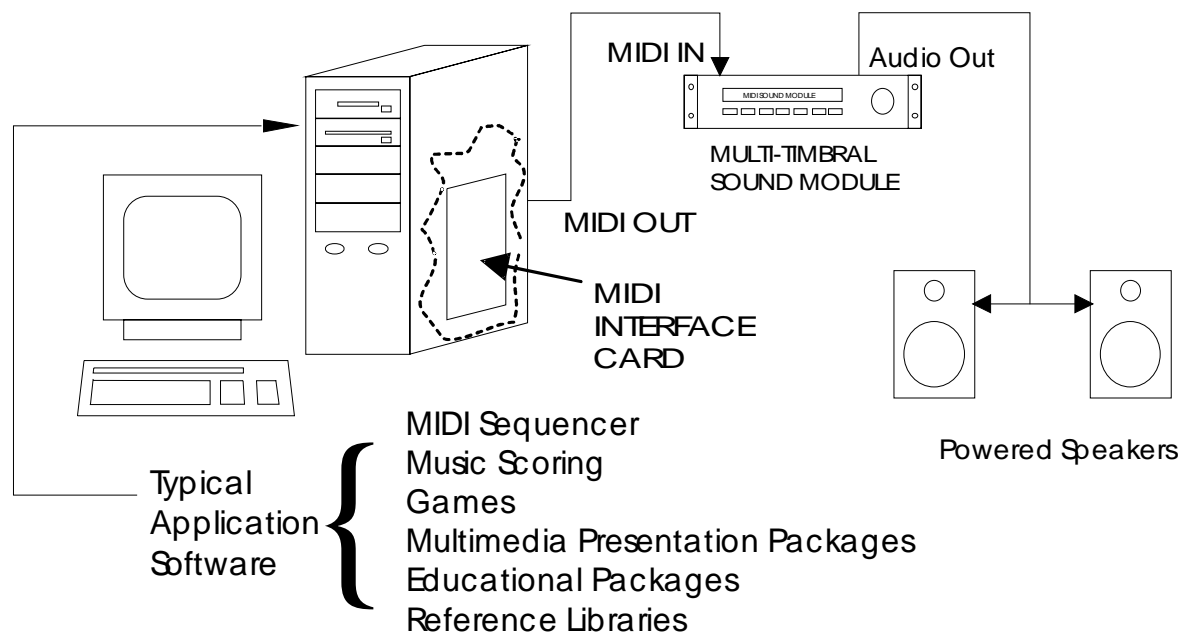
Figure 3. A PC-Based MIDI System

MIDI Messages

A MIDI message is made up of an eight-bit status byte which is generally followed by one or two data bytes. There are a number of different types of MIDI messages. At the highest level, MIDI messages are classified as being either Channel Messages or System Messages. Channel messages are those which apply to a specific Channel, and the Channel number is included in the status byte for these messages. System messages are not Channel specific, and no Channel number is indicated in their status bytes.

Channel Messages may be further classified as being either Channel Voice Messages, or Mode Messages. Channel Voice Messages carry musical performance data, and these messages comprise most of the traffic in a typical MIDI data stream. Channel Mode messages affect the way a receiving instrument will respond to the Channel Voice messages.

Channel Voice Messages

Channel Voice Messages are used to send musical performance information. The messages in this category are the Note On, Note Off, Polyphonic Key Pressure, Channel Pressure, Pitch Bend Change, Program Change, and the Control Change messages.

• Note On / Note Off / Velocity

In MIDI systems, the activation of a particular note and the release of the same note are considered as two separate events. When a key is pressed on a MIDI keyboard instrument or MIDI keyboard controller, the keyboard sends a Note On message on the MIDI OUT port. The keyboard may be set to transmit on any one of the sixteen logical MIDI channels, and the status byte for the Note On message will indicate the selected Channel number. The Note On status byte is followed by two data bytes, which specify key number (indicating which key was pressed) and velocity (how hard the key was pressed).

The key number is used in the receiving synthesizer to select which note should be played, and the velocity is normally used to control the amplitude of the note. When the key is released, the keyboard instrument or controller will send a Note Off message. The Note Off message also includes data bytes for the key number and for the velocity with which the key was released. The Note Off velocity information is normally ignored.

- Aftertouch

Some MIDI keyboard instruments have the ability to sense the amount of pressure which is being applied to the keys while they are depressed. This pressure information, commonly called "aftertouch", may be used to control some aspects of the sound produced by the synthesizer (vibrato, for example). If the keyboard has a pressure sensor for each key, then the resulting "polyphonic aftertouch" information would be sent in the form of Polyphonic Key Pressure messages. These messages include separate data bytes for key number and pressure amount. It is currently more common for keyboard instruments to sense only a single pressure level for the entire keyboard. This "Channel aftertouch" information is sent using the Channel Pressure message, which needs only one data byte to specify the pressure value.

- Pitch Bend

The Pitch Bend Change message is normally sent from a keyboard instrument in response to changes in position of the pitch bend wheel. The pitch bend information is used to modify the pitch of sounds being played on a given Channel. The Pitch Bend message includes two data bytes to specify the pitch bend value. Two bytes are required to allow fine enough resolution to make pitch changes resulting from movement of the pitch bend wheel seem to occur in a continuous manner rather than in steps.

- Program Change

The Program Change message is used to specify the type of instrument which should be used to play sounds on a given Channel. This message needs only one data byte which specifies the new program number.

- Control Change

MIDI Control Change messages are used to control a wide variety of functions in a synthesizer. Control Change messages, like other MIDI Channel messages, should only affect the Channel number indicated in the status byte. The Control Change status byte is followed by one data byte indicating the "controller number", and a second byte which specifies the "control value". The controller number identifies which function of the synthesizer is to be controlled by the message. A complete list of assigned controllers is found in the MIDI 1.0 Detailed Specification.

- Bank Select

Controller number zero (with 32 as the LSB) is defined as the bank select. The bank select function is used in some synthesizers in conjunction with the MIDI Program Change message to expand the number of different instrument sounds which may be specified (the Program Change message alone allows selection of one of 128 possible program numbers). The additional sounds are selected by preceding the Program Change message with a Control Change message which specifies a new value for Controller zero and Controller 32, allowing 16,384 banks of 128 sound each.

Since the MIDI specification does not describe the manner in which a synthesizer's banks are to be mapped to Bank Select messages, there is no standard way for a Bank Select message to select a specific synthesizer bank. Some manufacturers, such as Roland (with "GS") and

Yamaha (with "XG"), have adopted their own practices to assure some standardization within their own product lines.

- RPN / NRPN

Controller number 6 (Data Entry), in conjunction with Controller numbers 96 (Data Increment), 97 (Data Decrement), 98 (Non-Registered Parameter Number LSB), 99 (Non-Registered Parameter Number MSB), 100 (Registered Parameter Number LSB), and 101 (Registered Parameter Number MSB), extend the number of controllers available via MIDI. Parameter data is transferred by first selecting the parameter number to be edited using controllers 98 and 99 or 100 and 101, and then adjusting the data value for that parameter using controller number 6, 96, or 97.

RPN and NRPN are typically used to send parameter data to a synthesizer in order to edit sound patches or other data. Registered parameters are those which have been assigned some particular function by the MIDI Manufacturers Association (MMA) and the Japan MIDI Standards Committee (JMSC). For example, there are Registered Parameter numbers assigned to control pitch bend sensitivity and master tuning for a synthesizer. Non-Registered parameters have not been assigned specific functions, and may be used for different functions by different manufacturers. Here again, Roland and Yamaha, among others, have adopted their own practices to assure some standardization.

Channel Mode Messages

Channel Mode messages (MIDI controller numbers 121 through 127) affect the way a synthesizer responds to MIDI data. Controller number 121 is used to reset all controllers. Controller number 122 is used to enable or disable Local Control (In a MIDI synthesizer which has it's own keyboard, the functions of the keyboard controller and the synthesizer can be isolated by turning Local Control off). Controller numbers 124 through 127 are used to select between Omni Mode On or Off, and to select between the Mono Mode or Poly Mode of operation.

When Omni mode is On, the synthesizer will respond to incoming MIDI data on all channels. When Omni mode is Off, the synthesizer will only respond to MIDI messages on one Channel. When Poly mode is selected, incoming Note On messages are played polyphonically. This means that when multiple Note On messages are received, each note is assigned its own voice (subject to the number of voices available in the synthesizer). The result is that multiple notes are played at the same time. When Mono mode is selected, a single voice is assigned per MIDI Channel. This means that only one note can be played on a given Channel at a given time. Most modern MIDI synthesizers will default to Omni On/Poly mode of operation. In this mode, the synthesizer will play note messages received on any MIDI Channel, and notes received on each Channel are played polyphonically. In the Omni Off/Poly mode of operation, the synthesizer will receive on a single Channel and play the notes received on this Channel polyphonically. This mode could be useful when several synthesizers are daisy-chained using MIDI THRU. In this case each synthesizer in the chain can be set to play one part (the MIDI data on one Channel), and ignore the information related to the other parts.

Note that a MIDI instrument has one MIDI Channel which is designated as its "Basic Channel". The Basic Channel assignment may be hard-wired, or it may be selectable. Mode messages can only be received by an instrument on the Basic Channel.

System Messages

MIDI System Messages are classified as being System Common Messages, System Real Time Messages, or System Exclusive Messages. System Common messages are intended for all receivers in the system. System Real Time messages are used for synchronization between

clock-based MIDI components. System Exclusive messages include a Manufacturer's Identification (ID) code, and are used to transfer any number of data bytes in a format specified by the referenced manufacturer.

• System Common Messages

The System Common Messages which are currently defined include MTC Quarter Frame, Song Select, Song Position Pointer, Tune Request, and End Of Exclusive (EOX). The MTC Quarter Frame message is part of the MIDI Time Code information used for synchronization of MIDI equipment and other equipment, such as audio or video tape machines.

The Song Select message is used with MIDI equipment, such as sequencers or drum machines, which can store and recall a number of different songs. The Song Position Pointer is used to set a sequencer to start playback of a song at some point other than at the beginning. The Song Position Pointer value is related to the number of MIDI clocks which would have elapsed between the beginning of the song and the desired point in the song. This message can only be used with equipment which recognizes MIDI System Real Time Messages (MIDI Sync).

The Tune Request message is generally used to request an analog synthesizer to retune its' internal oscillators. This message is generally not needed with digital synthesizers.

The EOX message is used to flag the end of a System Exclusive message, which can include a variable number of data bytes.

• System Real Time Messages

The MIDI System Real Time messages are used to synchronize all of the MIDI clock-based equipment within a system, such as sequencers and drum machines. Most of the System Real Time messages are normally ignored by keyboard instruments and synthesizers. To help ensure accurate timing, System Real Time messages are given priority over other messages, and these single-byte messages may occur anywhere in the data stream (a Real Time message may appear between the status byte and data byte of some other MIDI message).

The System Real Time messages are the Timing Clock, Start, Continue, Stop, Active Sensing, and the System Reset message. The Timing Clock message is the master clock which sets the tempo for playback of a sequence. The Timing Clock message is sent 24 times per quarter note. The Start, Continue, and Stop messages are used to control playback of the sequence.

The Active Sensing signal is used to help eliminate "stuck notes" which may occur if a MIDI cable is disconnected during playback of a MIDI sequence. Without Active Sensing, if a cable is disconnected during playback, then some notes may be left playing indefinitely because they have been activated by a Note On message, but the corresponding Note Off message will never be received.

The System Reset message, as the name implies, is used to reset and initialize any equipment which receives the message. This message is generally not sent automatically by transmitting devices, and must be initiated manually by a user.

• System Exclusive Messages

System Exclusive messages may be used to send data such as patch parameters or sample data between MIDI devices. Manufacturers of MIDI equipment may define their own formats for System Exclusive data. Manufacturers are granted unique identification (ID) numbers by the MMA or the JMSC, and the manufacturer ID number is included as part of the System Exclusive message. The manufacturers ID is followed by any number of data bytes, and the

data transmission is terminated with the EOX message. Manufacturers are required to publish the details of their System Exclusive data formats, and other manufacturers may freely utilize these formats, provided that they do not alter or utilize the format in a way which conflicts with the original manufacturers specifications.

Certain System Exclusive ID numbers are reserved for special protocols. Among these are the MIDI Sample Dump Standard, which is a System Exclusive data format defined in the MIDI specification for the transmission of sample data between MIDI devices, as well as MIDI Show Control and MIDI Machine Control.

Timing Accuracy and Running Status

Since MIDI was designed to convey musical performance data, it must provide sufficiently accurate timing to preserve the rhythmic integrity of the music. The ear is quite sensitive to small variations in timing, which can easily degrade the expressive quality of a musical phrase. This is particularly true for grace notes, strummed chords, flams and clusters of notes, and for rhythmically complex and syncopated ensemble music.

Jitter (the variation in the relative timing between two or more events) has the strongest impact on rhythmic integrity. Various studies have shown that jitter on the close order of one ms can be audible. Other research has indicated that musicians can control relative time intervals with a precision of about 1.5 ms in common musical situations, as mentioned in the preceding paragraph.

Latency (the delay between when an event is triggered and when the resulting sound occurs) is also important: musical instruments feel more and more sluggish to play as latency increases. Since sound travels at about 1 ms per foot, latency of 7 ms is roughly equal to the maximum separation between members of a string quartet. In practice, latency of 10 ms is generally imperceptible, as long as the variation in latency (i.e. jitter) is kept small.

With a data transmission rate of 31.25 Kbit/s and 10 bits transmitted per byte of MIDI data, a 3-byte Note On or Note Off message takes about 1 ms to be sent. In practice, MIDI can often provide less than 1 ms jitter, with latency of 3 ms or less * as long as it's not necessary to actually play two events absolutely simultaneously. Since MIDI data is transmitted serially, a pair of musical events which originally occurred at the same time * but must be sent one at a time in the MIDI data stream * cannot be reproduced at exactly the same time. Luckily, human performers almost never play two notes at exactly the same time. For both biomechanical and expressive reasons, notes are generally spaced at least slightly apart. This allows MIDI to reproduce a solo musical part with quite reasonable rhythmic accuracy.

However, MIDI data being sent from a sequencer can include a number of different parts. On a given beat, there may be a large number of musical events which should occur virtually simultaneously. In this situation, many events will have to "wait their turn" to be transmitted over MIDI. Worse, different events will be delayed by different amounts of time (depending on how many events are queued up ahead of a given event). This can produce a kind of progressive rhythmic "smearing" that may be quite noticeable. A technique called "running status" is provided to help reduce this rhythmic "smearing" effect by reducing the amount of data actually transmitted in the MIDI data stream.

Running status is based on the fact that it is very common for a string of consecutive messages to be of the same message type. For instance, when a chord is played on a keyboard, ten successive Note On messages may be generated, followed by ten Note Off messages. When running status is used, a status byte is sent for a message only when the message is not of the same type as the last message sent on the same Channel. The status byte for subsequent messages of the same type may be omitted (only the data bytes are sent for these subsequent messages).

MIDI and Wavetable Synthesis Tutorial

The effectiveness of running status can be enhanced by sending Note On messages with a velocity of zero in place of Note Off messages. In this case, long strings of Note On messages will often occur. Changes in some of the MIDI controllers or movement of the pitch bend wheel on a musical instrument can produce a staggering number of MIDI Channel voice messages, and running status can also help a great deal in these instances.

MIDI Sequencers and Standard MIDI Files

MIDI messages are received and processed by a MIDI synthesizer in real time. When the synthesizer receives a MIDI "note on" message it plays the appropriate sound. When the corresponding "note off" message is received, the synthesizer turns the note off. If the source of the MIDI data is a musical instrument keyboard, then this data is being generated in real time. When a key is pressed on the keyboard, a "note on" message is generated in real time. In these real time applications, there is no need for timing information to be sent along with the MIDI messages.

However, if the MIDI data is to be stored as a data file, and/or edited using a sequencer, then some form of "time-stamping" for the MIDI messages is required. The Standard MIDI Files specification provides a standardized method for handling time-stamped MIDI data. This standardized file format for time-stamped MIDI data allows different applications, such as sequencers, scoring packages, and multimedia presentation software, to share MIDI data files.

The specification for Standard MIDI Files defines three formats for MIDI files. MIDI sequencers can generally manage multiple MIDI data streams, or "tracks". Standard MIDI files using Format 0 store all of the MIDI sequence data in a single track. Format 1 files store MIDI data as a collection of tracks. Format 2 files can store several independent patterns. Format 2 is generally not used by MIDI sequencers for musical applications. Most sophisticated MIDI sequencers can read either Format 0 or Format 1 Standard MIDI Files. Format 0 files may be smaller, and thus conserve storage space. They may also be transferred using slightly less system bandwidth than Format 1 files. However, Format 1 files may be viewed and edited more directly, and are therefore generally preferred.

Synthesizer Basics

Polyphony

The polyphony of a sound generator refers to its ability to play more than one note at a time. Polyphony is generally measured or specified as a number of notes or voices. Most of the early music synthesizers were monophonic, meaning that they could only play one note at a time. If you pressed five keys simultaneously on the keyboard of a monophonic synthesizer, you would only hear one note. Pressing five keys on the keyboard of a synthesizer which was polyphonic with four voices of polyphony would, in general, produce four notes. If the keyboard had more voices (many modern sound modules have 16, 24, or 32 note polyphony), then you would hear all five of the notes.

Sounds

The different sounds that a synthesizer or sound generator can produce are sometimes called "patches", "programs", "algorithms", or "timbres". Programmable synthesizers commonly assign "program numbers" (or patch numbers) to each sound. For instance, a sound module might use patch number 1 for its acoustic piano sound, and patch number 36 for its fretless bass sound. The association of all patch numbers to all sounds is often referred to as a patch map.

Via MIDI, a Program Change message is used to tell a device receiving on a given Channel to change the instrument sound being used. For example, a sequencer could set up devices on Channel 4 to play fretless bass sounds by sending a Program Change message for Channel four with a data byte value of 36 (this is the General MIDI program number for the fretless bass patch).

Multitimbral Mode

A synthesizer or sound generator is said to be multitimbral if it is capable of producing two or more different instrument sounds simultaneously. If a synthesizer can play five notes simultaneously, and it can produce a piano sound and an acoustic bass sound at the same time, then it is multitimbral. With enough notes of polyphony and "parts" (multitimbral) a single synthesizer could produce the entire sound of a band or orchestra.

Multitimbral operation will generally require the use of a sequencer to send the various MIDI messages required. For example, a sequencer could send MIDI messages for a piano part on Channel 1, bass on Channel 2, saxophone on Channel 3, drums on Channel 10, etc. A 16 part multitimbral synthesizer could receive a different part on each of MIDI's 16 logical channels.

The polyphony of a multitimbral synthesizer is usually allocated dynamically among the different parts (timbres) being used. At any given instant five voices might be needed for the piano part, two voices for the bass, one for the saxophone, plus 6 voices for the drums. Note that some sounds on some synthesizers actually utilize more than one "voice", so the number of notes which may be produced simultaneously may be less than the stated polyphony of the synthesizer, depending on which sounds are being utilized.

The General MIDI (GM) System

At the beginning of a MIDI sequence, a Program Change message is usually sent on each Channel used in the piece in order to set up the appropriate instrument sound for each part. The Program Change message tells the synthesizer which patch number should be used for a particular MIDI Channel. If the synthesizer receiving the MIDI sequence uses the same patch map (the assignment of patch numbers to sounds) that was used in the composition of the sequence, then the sounds will be assigned as intended.

Prior to General MIDI, there was no standard for the relationship of patch numbers to specific sounds for synthesizers. Thus, a MIDI sequence might produce different sounds when played on different synthesizers, even though the synthesizers had comparable types of sounds. For example, if the composer had selected patch number 5 for Channel 1, intending this to be an electric piano sound, but the synthesizer playing the MIDI data had a tuba sound mapped at patch number 5, then the notes intended for the piano would be played on the tuba when using this synthesizer (even though this synthesizer may have a fine electric piano sound available at some other patch number).

The General MIDI (GM) Specification defines a set of general capabilities for General MIDI Instruments. The General MIDI Specification includes the definition of a General MIDI Sound Set (a patch map), a General MIDI Percussion map (mapping of percussion sounds to note numbers), and a set of General MIDI Performance capabilities (number of voices, types of MIDI messages recognized, etc.). A MIDI sequence which has been generated for use on a General MIDI Instrument should play correctly on any General MIDI synthesizer or sound module. The General MIDI system utilizes MIDI Channels 1-9 and 11-16 for chromatic instrument sounds, while Channel number 10 is utilized for "key-based" percussion sounds. These instrument sounds are grouped into "sets" of related sounds. For example, program numbers 1-8 are piano sounds, 9-16 are chromatic percussion sounds, 17-24 are organ sounds, 25-32 are guitar sounds, etc.

For the instrument sounds on channels 1-9 and 11-16, the note number in a Note On message is used to select the pitch of the sound which will be played. For example if the Vibraphone instrument (program number 12) has been selected on Channel 3, then playing note number 60 on Channel 3 would play the middle C note (this would be the default note to pitch assignment on most instruments), and note number 59 on Channel 3 would play B below middle C. Both notes would be played using the Vibraphone sound.

The General MIDI percussion sounds are set on Channel 10. For these "key-based" sounds, the note number data in a Note On message is used differently. Note numbers on Channel 10 are used to select which drum sound will be played. For example, a Note On message on Channel 10 with note number 60 will play a Hi Bongo drum sound. Note number 59 on Channel 10 will play the Ride Cymbal 2 sound.

It should be noted that the General MIDI system specifies sounds using program numbers 1 through 128. The MIDI Program Change message used to select these sounds uses an 8-bit byte, which corresponds to decimal numbering from 0 through 127, to specify the desired program number. Thus, to select GM sound number 10, the Glockenspiel, the Program Change message will have a data byte with the decimal value 9.

The General MIDI system specifies which instrument or sound corresponds with each program/patch number, but General MIDI does not specify how these sounds are produced. Thus, program number 1 should select the Acoustic Grand Piano sound on any General MIDI instrument. However, the Acoustic Grand Piano sound on two General MIDI synthesizers which use different synthesis techniques may sound quite different.

Synthesis Technology: FM and Wavetable

There are a number of different technologies or algorithms used to create sounds in music synthesizers. Two widely used techniques are Frequency Modulation (FM) synthesis and Wavetable synthesis.

FM synthesis techniques generally use one periodic signal (the modulator) to modulate the frequency of another signal (the carrier). If the modulating signal is in the audible range, then the result will be a significant change in the timbre of the carrier signal. Each FM voice requires a minimum of two signal generators. These generators are commonly referred to as "operators", and different FM synthesis implementations have varying degrees of control over the operator parameters. Sophisticated FM systems may use 4 or 6 operators per voice, and the operators may have adjustable envelopes which allow adjustment of the attack and decay rates of the signal. Although FM systems were implemented in the analog domain on early synthesizer keyboards, modern FM synthesis implementations are done digitally. FM synthesis techniques are very useful for creating expressive new synthesized sounds. However, if the goal of the synthesis system is to recreate the sound of some existing instrument, this can generally be done more accurately with digital sample-based techniques.

Digital sampling systems store high quality sound samples digitally, and then replay these sounds on demand. Digital sample-based synthesis systems may employ a variety of special techniques, such as sample looping, pitch shifting, mathematical interpolation, and digital filtering, in order to reduce the amount of memory required to store the sound samples (or to get more types of sounds from a given amount of memory). These sample-based synthesis systems are often called "wavetable" synthesizers (the sample memory in these systems contains a large number of sampled sound segments, and can be thought of as a "table" of sound waveforms which may be looked up and utilized when needed).

Wavetable Synthesis Techniques

The majority of professional synthesizers available today use some form of sampled-sound or Wavetable synthesis. The trend for multimedia sound products is also towards wavetable synthesis. To help prospective MIDI developers, a number of the techniques employed in this type of synthesis are discussed in the following paragraphs.

• Looping and Envelope Generation

One of the primary techniques used in wavetable synthesizers to conserve sample memory space is the looping of sampled sound segments. For many instrument sounds, the sound can be modeled as consisting of two major sections: the attack section and the sustain section. The attack section is the initial part of the sound, where the amplitude and the spectral characteristics of the sound may be changing very rapidly. The sustain section of the sound is that part of the sound following the attack, where the characteristics of the sound are changing less dynamically.

Figure 4 shows a waveform with portions which could be considered the attack and the sustain sections indicated.   In this example, the spectral characteristics of the waveform remain constant throughout the sustain section, while the amplitude is decreasing at a fairly constant rate. This is an exaggerated example, in most natural instrument sounds, both the spectral characteristics and the amplitude continue to change through the duration of the sound. The sustain section, if one can be identified, is that section for which the characteristics of the sound are relatively constant.

Attack Section   Sustain Section
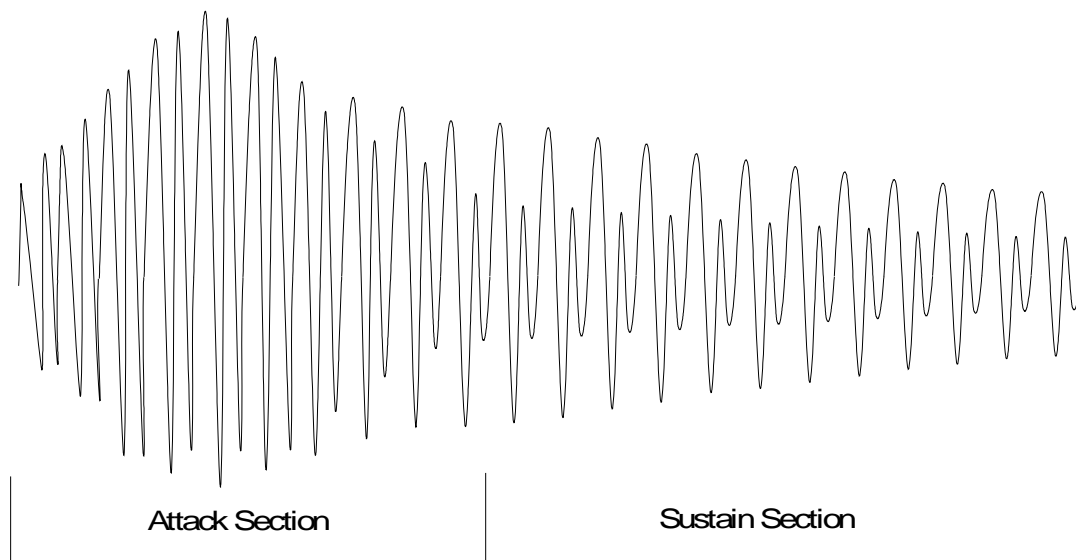
Figure 4. Attack and Sustain Portions of a Waveform

A great deal of memory can be saved in wavetable synthesis systems by storing only a short segment of the sustain section of the waveform, and then looping this segment during playback. Figure 5 shows a two period segment of the sustain section from the waveform in Figure 4, which has been looped to create a steady state signal. If the original sound had a fairly constant spectral content and amplitude during the sustained section, then the sound resulting from this looping operation should be a good approximation of the sustained section of the original.

2 period segment
of sustain sound

Waveform resulting from
looping segment above
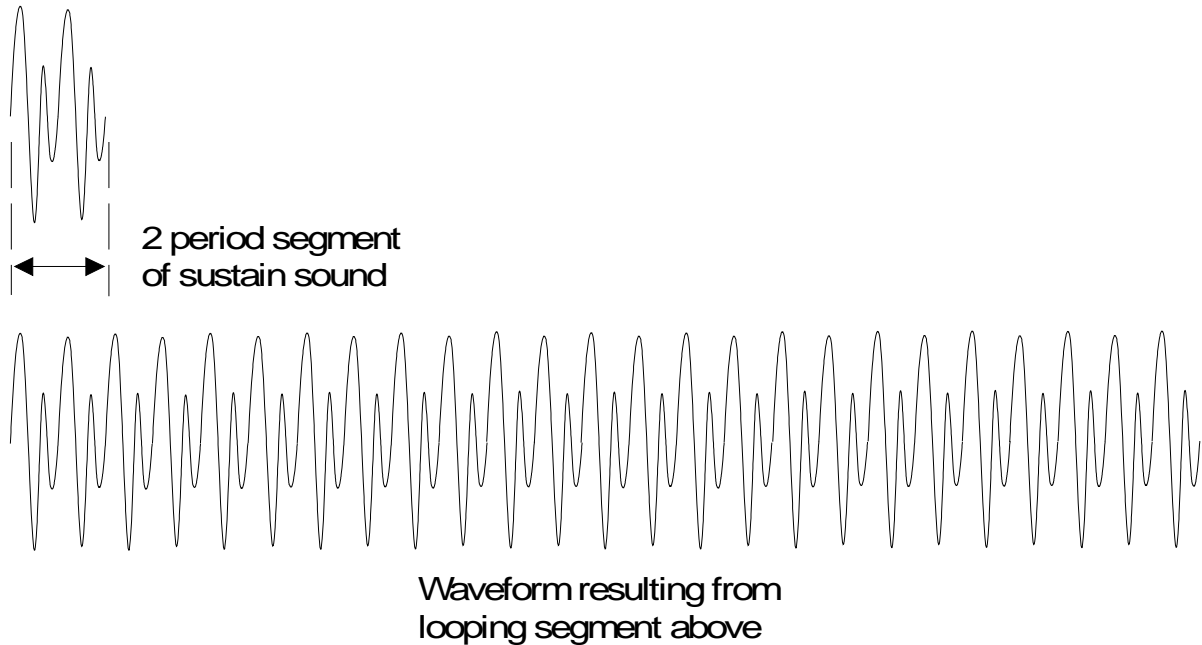
Figure 5. Looping of a Sample Segment

For many acoustic string instruments, the spectral characteristics of the sound remain fairly constant during the sustain section, while the amplitude of the signal decays. This can be simulated with a looped segment by multiplying the looped samples by a decreasing gain factor during playback to get the desired shape or envelope. The amplitude envelope of a sound is commonly modeled as consisting of some number of linear segments. An example is the commonly used four part piecewise-linear Attack-Decay-Sustain-Release (ADSR) envelope model. Figure 6 depicts a typical ADSR envelope shape, and Figure 7 shows the result of applying this envelope to the looped waveform from Figure 5.
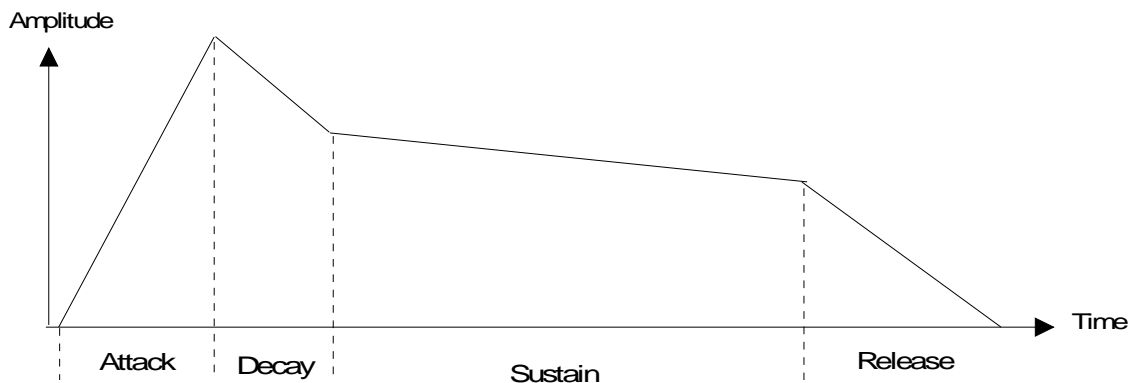
Figure 6. A Typical ADSR Amplitude Envelope
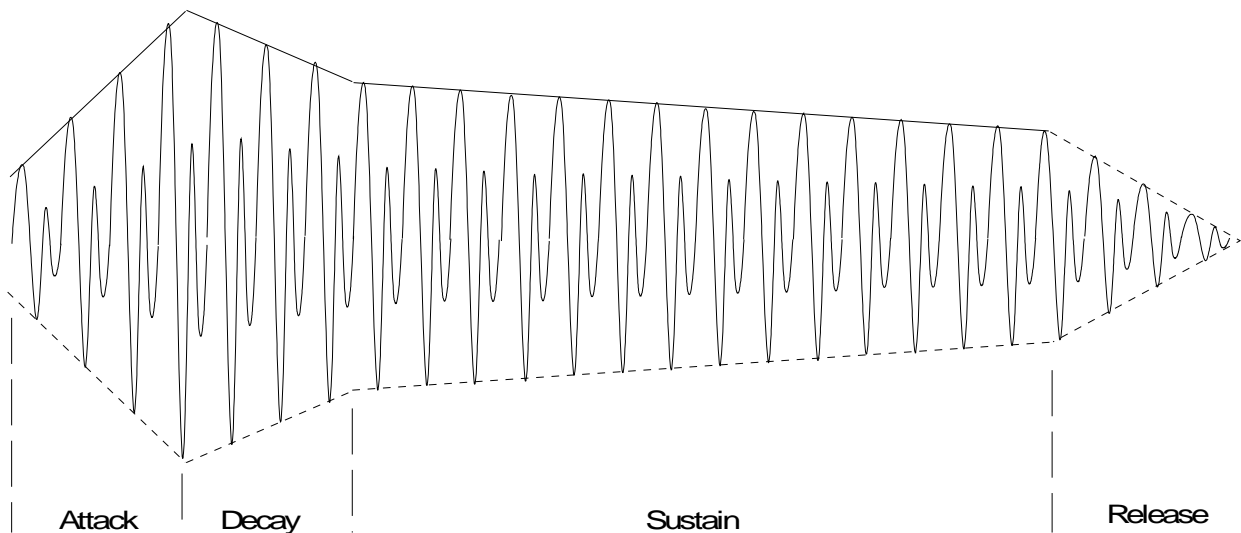
Figure 7. ADSR Envelope Applied to a Looped Sample Segment

A typical wavetable synthesis system would store sample data for the attack section and the looped section of an instrument sound. These sample segments might be referred to as the initial sound and the loop sound. The initial sound is played once through, and then the loop sound is played repetitively until the note ends. An envelope generator function is used to create an envelope which is appropriate for the particular instrument, and this envelope is applied to the output samples during playback.

Playback of the initial wave (with the attack portion of the envelope applied) begins when a Note On message is received. The length of the initial sound segment is fixed by the number of samples in the segment, and the length of the attack and decay sections of the envelope are generally also fixed for a given instrument sound.

The sustain section will continue to repeat the loop samples while applying the sustain envelope slope (which decays slowly in our examples), until a Note Off message is applied. The Note Off message triggers the beginning of the release portion of the envelope.

• Loop Length

The loop length is measured as a number of samples, and the length of the loop should be equal to an integral number of periods of the fundamental pitch of the sound being played (if this is not true, then an undesirable "pitch shift" will occur during playback when the looping begins). In practice, the length of the loop segment for an acoustic instrument sample may be many periods with respect to the fundamental pitch of the sound. If the sound has a natural vibrato or chorus effect, then it is generally desirable to have the loop segment length be an integral multiple of the period of the vibrato or chorus.

• One-Shot Sounds

The previous paragraphs discussed dividing a sampled sound into an attack section and a sustain section, and then using looping techniques to minimize the storage requirements for the sustain portion. However, some sounds, particularly sounds of short duration or sounds whose characteristics change dynamically throughout their duration, are not suitable for looped playback techniques. Short drum sounds often fit this description. These sounds are

stored as a single sample segment which is played once through with no looping. This class of sounds are referred to as "one-shot" sounds.

• Sample Editing and Processing

There are a number of sample editing and processing steps involved in preparing sampled sounds for use in a wavetable synthesis system. The requirements for editing the original sample data to identify and extract the initial and loop segments have already been mentioned.

Editing may also be required to make the endpoints of the loop segment compatible. If the amplitude and the slope of the waveform at the beginning of the loop segment do not match those at the end of the loop, then a repetitive "glitch" will be heard during playback of the looped section. Additional processing may be performed to "compress" the dynamic range of the sound to improve the signal/quantizing noise ratio or to conserve sample memory. This topic is addressed next.

When all of the sample processing has been completed, the resulting sampled sound segments for the various instruments are tabulated to form the sample memory for the synthesizer.

• Sample Data Compression

The signal-to-quantizing noise ratio for a digitally sampled signal is limited by sample word size (the number of bits per sample), and by the amplitude of the digitized signal. Most acoustic instrument sounds reach their peak amplitude very quickly, and the amplitude then slowly decays from this peak. The ear's sensitivity dynamically adjusts to signal level. Even in systems utilizing a relatively small sample word size, the quantizing noise level is generally not perceptible when the signal is near maximum amplitude. However, as the signal level decays, the ear becomes more sensitive, and the noise level will appear to increase. Of course, using a larger word size will reduce the quantizing noise, but there is a considerable price penalty paid if the number of samples is large.

Compression techniques may be used to improve the signal-to-quantizing noise ratio for some sampled sounds. These techniques reduce the dynamic range of the sound samples stored in the sample memory. The sample data is decompressed during playback to restore the dynamic range of the signal. This allows the use of sample memory with a smaller word size (smaller dynamic range) than is utilized in the rest of the system. There are a number of different compression techniques which may be used to compress the dynamic range of a signal.

Note that there is some compression effect inherent in the looping techniques described earlier. If the loop segment is stored at an amplitude level which makes full use of the dynamic range available in the sample memory, and the processor and D/A converters used for playback have a wider dynamic range than the sample memory, then the application of a decay envelope during playback will have a decompression effect similar to that described in the previous paragraph.

• Pitch Shifting

In order to minimize sample memory requirements, wavetable synthesis systems utilize pitch shifting, or pitch transposition techniques, to generate a number of different notes from a single sound sample of a given instrument. For example, if the sample memory contains a sample of a middle C note on the acoustic piano, then this same sample data could be used to generate the C# note or D note above middle C using pitch shifting.

Pitch shifting is accomplished by accessing the stored sample data at different rates during playback. For example, if a pointer is used to address the sample memory for a sound, and the

pointer is incremented by one after each access, then the samples for this sound would be accessed sequentially, resulting in some particular pitch. If the pointer increment was two rather than one, then only every second sample would be played, and the resulting pitch would be shifted up by one octave (the frequency would be doubled).

In the previous example, the sample memory address pointer was incremented by an integer number of samples. This allows only a limited set of pitch shifts. In a more general case, the memory pointer would consist of an integer part and a fractional part, and the increment value could be a fractional number of samples. The memory pointer is often referred to as a "phase accumulator" and the increment value is then the "phase increment". The integer part of the phase accumulator is used to address the sample memory, the fractional part is used to maintain frequency accuracy.

For example if the phase increment value was equivalent to 1/2, then the pitch would be shifted down by one octave (the frequency would be halved). A phase increment value of 1.05946 (the twelfth root of two) would create a pitch shift of one musical half-step (i.e. from C to C#) compared with an increment of 1. When non-integer increment values are utilized, the frequency resolution for playback is determined by the number of bits used to represent the fractional part of the address pointer and the address increment parameter.

• Interpolation

When the fractional part of the address pointer is non-zero, then the "desired value" falls between available data samples. Figure 8 depicts a simplified addressing scheme wherein the Address Pointer and the increment parameter each have a 4-bit integer part and a 4-bit fractional part. In this case, the increment value is equal to 1 1/2 samples. Very simple systems might simply ignore the fractional part of the address when determining the sample value to be sent to the D/A converter. The data values sent to the D/A converter when using this approach are indicated in the Figure 8, case I (next page).

A slightly better approach would be to use the nearest available sample value. More sophisticated systems would perform some type of mathematical interpolation between available data points in order to get a value to be used for playback. Values which might be sent to the D/A when interpolation is employed are shown as case II. Note that the overall frequency accuracy would be the same for both cases indicated, but the output is severely distorted in the case where interpolation is not used.

There are a number of different algorithms used for interpolation between sample values. The simplest is linear interpolation. With linear interpolation, interpolated value is simply the weighted average of the two nearest samples, with the fractional address used as a weighting constant. For example, if the address pointer indicated an address of $(n+K)$, where n is the integer part of the address and K is the fractional part, than the interpolated value can be calculated as $s(n+K) = (1-K)s(n) + (K)s(n+1)$, where $s(n)$ is the sample data value at address n. More sophisticated interpolation techniques can be utilized to further reduce distortion, but these techniques are computationally expensive.
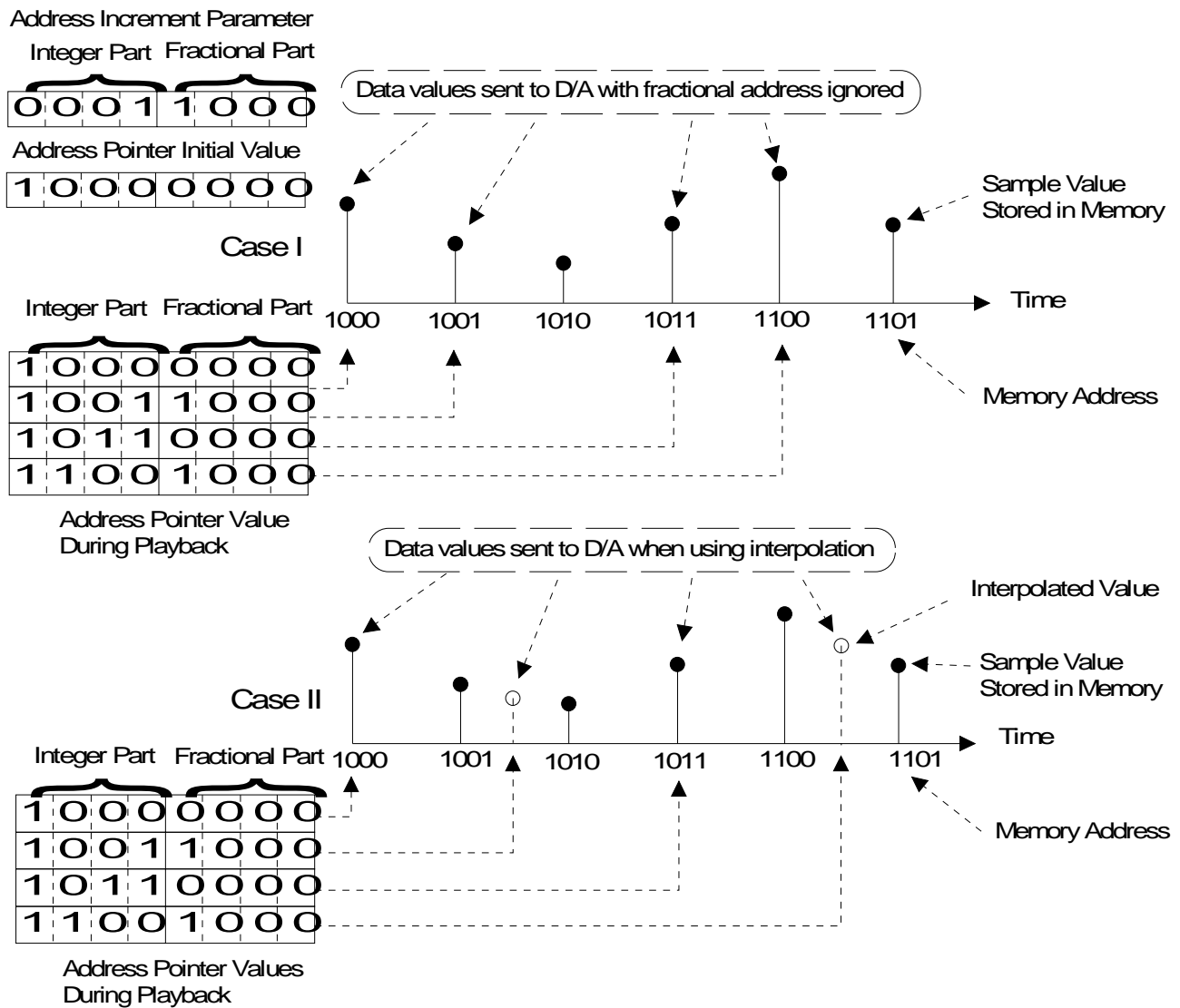
Figure 8. Sample Memory Addressing and Interpolation

• Oversampling

Oversampling of the sound samples may also be used to improve distortion in wavetable synthesis systems. For example, if 4X oversampling were utilized for a particular instrument sound sample, then an address increment value of 4 would be used for playback with no pitch shift. The data points chosen during playback will be closer to the "desired values", on the average, than they would be if no oversampling were utilized because of the increased number of data points used to represent the waveform. Of course, oversampling has a high cost in terms of sample memory requirements.

In many cases, the best approach may be to utilize linear interpolation combined with varying degrees of oversampling where needed. The linear interpolation technique provides reasonable accuracy for many sounds, without the high penalty in terms of processing power required for more sophisticated interpolation methods. For those sounds which need better accuracy, oversampling is employed. With this approach, the additional memory required for oversampling is only utilized where it is most needed. The combined effect of linear interpolation and selective oversampling can produce excellent results.

- Splits

When the pitch of a sampled sound is changed during playback, the timbre of the sound is changed somewhat also. The effect is less noticeable for small changes in pitch (up to a few semitones), than it is for a large pitch shift. To retain a natural sound, a particular sample of an instrument sound will only be useful for recreating a limited range of notes. To get coverage of the entire instrument range, a number of different samples, each with a limited range of notes, are used. The resulting instrument implementation is often referred to as a "multisampled" instrument. This technique can be thought of as splitting a musical instrument keyboard into a number of ranges of notes, with a different sound sample used for each range. Each of these ranges is referred to as a split, or key split.

Velocity splits refer to the use of different samples for different note velocities. Using velocity splits, one sample might be utilized if a particular note is played softly, where a different sample would be utilized for the same note of the same instrument when played with a higher velocity. This technique is not commonly used to produce basic sound samples because of the added memory expense, but both key splitting and velocity splitting techniques can be utilized as a performance enhancement. For instance, a key split might allow a fretless bass sound on the lower octaves of a keyboard, while the upper octaves play a vibraphone. Similarly, a velocity split might "layer" strings on top of an acoustic piano sound when the keys are hit with higher velocity.

- Aliasing Noise

Earlier paragraphs discussed the timbre changes which result from pitch shifting. The resampling techniques used to shift the pitch of a stored sound sample can also result in the introduction of aliasing noise into an instrument sound. The generation of aliasing noise can also limit the amount of pitch shifting which may be effectively applied to a sound sample. Sounds which are rich in upper harmonic content will generally have more of a problem with aliasing noise. Low-pass filtering applied after interpolation can help eliminate the undesirable effect of aliasing noise. The use of oversampling also helps eliminate aliasing noise.

- LFOs for Vibrato and Tremolo

Vibrato and tremolo are effects which are often produced by musicians playing acoustic instruments. Vibrato is basically a low-frequency modulation of the pitch of a note, while tremolo is modulation of the amplitude of the sound. These effects are simulated in synthesizers by implementing low-frequency oscillators (LFOs) which are used to modulate the pitch or amplitude of the synthesized sound being produced.

Natural vibrato and tremolo effects tend to increase in strength as a note is sustained. This is accomplished in synthesizers by applying an envelope generator to the LFO. For example, a flute sound might have a tremolo effect which begins at some point after the note has sounded, and the tremolo effect gradually increases to some maximum level, where it remains until the note stops sounding.

- Layering

Layering refers to a technique in which multiple sounds are utilized for each note played. This technique can be used to generate very rich sounds, and may also be useful for increasing the number of instrument patches which can be created from a limited sample set. Note that layered sounds generally utilize more than one voice of polyphony for each note played, and thus the number of voices available is effectively reduced when these sounds are being used.

• Digital Filtering

It was mentioned earlier that low-pass filtering may be used to help eliminate noise which may be generated during the pitch shifting process. There are also a number of ways in which digital filtering is used in the timbre generation process to improve the resulting instrument sound. In these applications, the digital filter implementation is polyphonic, meaning that a separate filter is implemented for each voice being generated, and the filter implementation should have dynamically adjustable cutoff frequency and/or Q.

For many acoustic instruments, the character of the tone which is produced changes dramatically as a function of the amplitude level at which the instrument is played. For example, the tone of an acoustic piano may be very bright when the instrument is played forcefully, but much more mellow when it is played softly. Velocity splits, which utilize different sample segments for different note velocities, can be implemented to simulate this phenomena.

Another very powerful technique is to implement a digital low-pass filter for each note with a cutoff frequency which varies as a function of the note velocity. This polyphonic digital filter dynamically adjusts the output frequency spectrum of the synthesized sound as a function of note velocity, allowing a very effective recreation of the acoustic instrument timbre.

Another important application of digital filtering is in smoothing out the transitions between samples in key-based splits. At the border between two splits, there will be two adjacent notes which are based on different samples. Normally, one of these samples will have been pitch shifted up to create the required note, while the other will have been shifted down in pitch. As a result, the timbre of these two adjacent notes may be significantly different, making the split obvious. This problem may be alleviated by employing a digital filter which uses the note number to control the filter characteristics. A table may be constructed containing the filter characteristics for each note number of a given instrument. The filter characteristics are chosen to compensate for the pitch shifting associated with the key splits used for that instrument.

It is also common to control the characteristics of the digital filter using an envelope generator or an LFO. The result is an instrument timbre which has a spectrum which changes as a function of time. An envelope generator might be used to control the filter cutoff frequency generate a timbre which is very bright at the onset, but which gradually becomes more mellow as the note decays. Sweeping the cutoff frequency of a filter with a high Q setting using an envelope generator or LFO can help when trying to simulate the sounds of analog synthesizers.

The PC to MIDI Connection

To use MIDI with a personal computer, a PC to MIDI interface product is required unless the computer comes equipped with a built-in MIDI interface). The most common types of MIDI interfaces for IBM compatibles are built into the PC sound cards, sharing the function of the joystick port. However, many of these interfaces do not meet the MIDI specification for hardware ground isolation and may not be suitable for serious use. An alternative are add-in cards which plug into an expansion slot on the PC bus, but there are also serial port MIDI interfaces (connects to a serial port on the PC) and parallel port MIDI interfaces (connects to the PC printer port) and USB MIDI interfaces.

The fundamental function of a MIDI interface for the PC is to convert parallel data bytes from the PC data bus into the serial MIDI data format and vice versa (a UART function). However, "smart" MIDI interfaces may provide a number of more sophisticated functions, such as generation of MIDI timing data, MIDI data buffering, MIDI message filtering, synchronization to external tape machines, as well as multiple MIDI ports and more.

The specifics of the interface can be important to insure transparent operation of games and other applications which use General MIDI. GM does not define how the game is supposed to connect with the synthesizer, so sound-card and interface standards are also needed to assure proper operation. For the most part, modern-day operating systems provide a translation layer which assures compatibility among popular devices and software, but this is not true for older versions of Windows and for most MS-DOS software.

The defacto standard for MIDI interface add-in cards for the IBM-PC is the Roland MPU-401 interface. The MPU-401 is a smart MIDI interface, which also supports a dumb mode of operation (often referred to as "UART mode"). There are a number of MPU-401 compatible MIDI interfaces on the market, some which only support the UART (dumb) mode of operation. In addition, many IBM-PC add-in sound cards include built-in MIDI interfaces which implement the UART mode functions of the MPU-401.

PC Compatibility Issues

There are two levels of compatibility which must be considered for MIDI applications running on the PC. First is the compatibility of the application with the MIDI interface being used. The second is the compatibility of the application with the MIDI synthesizer. For the purposes of this tutorial we will be talking only about IBM-PC and compatible systems, though much of this information may also be applicable to other PC systems.

• MS-DOS Applications

If you have any older MS-DOS games you like to play, there is a good chance they have a MIDI sound track. If so, they probably support the MPU-401 interface, and most only require the UART mode – these applications should work correctly on any compatible PC equipped with a MPU-401 interface or a sound card with a MPU-401 UART-mode capability. Other MIDI interfaces, such as serial port or parallel port MIDI adapters, will only work if the application provides support for that particular model of MIDI interface.

Some MS-DOS applications provide support for a variety of different synthesizer modules, but most support only an FM synthesizer (AdLib/Sound Blaster), the MT-32 family from Roland, or General MIDI. Prior to the General MIDI standard, there was no widely accepted standard patch set for synthesizers, so applications generally needed to provide support for each of the most popular synthesizers at the time. If the application did not support the particular model of synthesizer or sound module that was attached to the PC, then the sounds produced by the application might not be the sounds which were intended. Modern applications can provide support for a General MIDI (GM) synthesizer, and any GM-compatible sound source should produce the correct sounds.

• Multimedia PC (MPC) Systems

In the early 90's the number of applications for high quality audio functions on the PC (including music synthesis) grew explosively after the introduction of Microsoft Windows 3.0 with Multimedia Extensions ("Windows with Multimedia") in 1991, characterized by the Multimedia PC (MPC) Specification. The audio capabilities of an MPC system include digital audio recording and playback (linear PCM sampling), music synthesis, and audio mixing.

The first MPC specification attempted to balance performance and cost issues by defining two types of synthesizers; a "Base Multitimbral Synthesizer", and an "Extended Multitimbral Synthesizer". Both the Base and the Extended synthesizer are expected to use a General MIDI patch set, but neither actually meets the full requirements of General MIDI polyphony or simultaneous timbres. Base Multitimbral Synthesizers must be capable of playing 6 "melodic notes" and "2 percussive" notes simultaneously, using 3 "melodic timbres" and 2 "percussive timbres".

The formal requirements for an Extended Multitimbral Synthesizer are only that it must have capabilities which exceed those specified for a Base Multitimbral Synthesizer. However, the "goals" for an Extended synthesizer include the ability to play 16 melodic notes and 8 percussive notes simultaneously, using 9 melodic timbres and 8 percussive timbres.

The MPC specification also included an authoring standard for MIDI composition. This standard required that each MIDI file contain two arrangements of the same song, one for Base synthesizers and one for Extended synthesizers, allowing for differences in available polyphony and timbres. The MIDI data for the Base synthesizer arrangement is sent on MIDI channels 13 - 16 (with the percussion track on Channel 16), and the Extended synthesizer arrangement utilizes channels 1 - 10 (percussion is on Channel 10). This technique was intended to optimize the MIDI file to play on both types of synthesizer, but required users of General MIDI synthesizers to disable certain Channels to avoid playing both performances, (including playing the Channel 16 percussion track, but with a melodic instrument). Microsoft eventually moved to full General MIDI support in Windows 95 and removed the Base/Extended requirement.

• Microsoft Windows 3.x/9x Configuration

Windowsapplications address hardware devices such as MIDI interfaces or synthesizers through the use of drivers. The drivers provide applications software with a common interface through which hardware may be accessed, and this simplifies the hardware compatibility issue. When a MIDI interface or synthesizer is installed in the PC and a suitable device driver has been loaded in Windows 3.x or 9x, the Windows MIDI Mapper applet will then appear within the Control Panel (Multimedia Control). MIDI messages are sent from an application to the MIDI Mapper, which then routes the messages to the appropriate device driver. The MIDI Mapper may be set to perform some filtering or translations of the MIDI messages in route from the application to the driver. The processing to be performed by the MIDI Mapper is defined in the MIDI Mapper Setups, Patch Maps, and Key Maps.

MIDI Mapper Setups are used to assign MIDI channels to device drivers. For instance, If you have an MPU-401 interface with a General MIDI synthesizer and you also have a Creative Labs Sound Blaster card in your system, you might wish to assign channels 13 to 16 to the Ad Lib driver (which will drive the Base-level FM synthesizer on the Sound Blaster), and assign channels 1 - 10 to the MPU-401 driver. In this case, MPC compatible MIDI files will play on both the General MIDI synthesizer and the FM synthesizer at the same time. The General MIDI synthesizer will play the Extended arrangement on MIDI channels 1 - 10, and the FM synthesizer will play the Base arrangement on channels 13-16.

The MIDI Mapper Setups can also be used to change the Channel number of MIDI messages. If you have MIDI files which were composed for a General MIDI instrument, and you are playing them on a Base Multitimbral Synthesizer, you would probably want to take the MIDI percussion data coming from your application on Channel 10 and send this information to the device driver on Channel 16.

The MIDI Mapper patch maps are used to translate patch numbers when playing MPC or General MIDI files on synthesizers which do not use the General MIDI patch numbers. Patch maps can also be used to play MIDI files which were arranged for non-GM synthesizers on GM synthesizers. For example, the Windows-supplied MT-32 patch map can be used when playing GM-compatible .MID files on the Roland MT-32 sound module or LAPC-1 sound card. The MIDI Mapper key maps perform a similar function, translating the key numbers contained in MIDI Note On and Note Off messages. This capability is useful for translating GM-compatible percussion parts for playback on non-GM synthesizers or vice-versa. The Windows-supplied MT-32 key map changes the key-to-drum sound assignments used for General MIDI to those used by the MT-32 and LAPC-1.

Summary

The MIDI protocol provides an efficient format for conveying musical performance data, and the Standard MIDI Files specification ensures that different applications can share time-stamped MIDI data. While this alone is largely sufficient for the working MIDI musician, the storage efficiency and on-the-fly editing capability of MIDI data also makes MIDI an attractive vehicle for generation of sounds in multimedia applications, computer games, or high-end karaoke equipment.

The General MIDI system provides a common set of capabilities and a common patch map for high polyphony, multitimbral synthesizers, providing musical sequence authors and multimedia applications developers with a common target platform for synthesis. With the greater realism which comes from wavetable synthesis, and as newer, interactive, applications come along, MIDI-driven synthesizers will continue to be an important component for sound generation devices and multimedia applications.

MIDI 1.0 Detailed Specification

Document Version 4.2
Revised February 1996

This document is a combination of the
MIDI 1.0 Detailed Specification v 4.1.1 and the
MIDI 1.0 Addendum v 4.2.
The MIDI Time Code Specification is not included.

Revised February 1996

MMA
POB 3173
La Habra CA 90632-3173

MIDI 1.0 Detailed Specification

TABLE OF CONTENTS

* Specification document available separately (see Table VIII).

APPENDIX

TABLES

INTRODUCTION

MIDI, the Musical Instrument Digital Interface, was established as a hardware and software specification which would make it possible to exchange information (musical notes, program changes, expression control, etc.) between different musical instruments or other devices such as sequencers, computers, lighting controllers, mixers, etc. This ability to transmit and receive data was originally conceived for live performances, although subsequent developments have had enormous impact in recording studios, audio and video production, and composition environments.

This document has been prepared as a joint effort between the MIDI Manufacturers Association (MMA) and the Japan MIDI Standards Committee (JMSC) to explain the MIDI 1.0 specification. This document is subject to change by agreement between the JMSC and MMA. Additional MIDI protocol may be included in supplements to this publication.

HARDWARE

The hardware MIDI interface operates at 31.25 (+/- 1%) Kbaud, asynchronous, with a start bit, 8 data bits (D0 to D7), and a stop bit. This makes a total of 10 bits for a period of 320 microseconds per serial byte. The start bit is a logical 0 (current on) and the stop bit is a logical 1 (current off). Bytes are sent LSB first.

Circuit: *(See Schematic - Page 2)*. 5 mA current loop type. Logical 0 is current ON. One output shall drive one and only one input. To avoid ground loops, and subsequent data errors, the transmitter circuitry and receiver circuitry are internally separated by an opto-isolator (a light emitting diode and a photo sensor which share a single, sealed package). Sharp PC-900 and HP 6N138 opto-isolators have been found acceptable. Other high-speed opto-isolators may be satisfactory. The receiver must require less than 5 mA to turn on. Rise and fall times should be less than 2 microseconds.

Connectors: DIN 5 pin (180 degree) female panel mount receptacle. An example is the SWITCHCRAFT 57 GB5F. The connectors shall be labeled "MIDI IN" and "MIDI OUT". Note that pins 1 and 3 are not used, and should be left unconnected in the receiver and transmitter. Pin 2 of the MIDI In connector should also be left unconnected.

The grounding shield connector on the MIDI jacks should not be connected to any circuit or chassis ground.

When MIDI Thru information is obtained from a MIDI In signal, transmission may occasionally be performed incorrectly due to signal degradation (caused by the response time of the opto-isolator) between the rising and falling edges of the square wave. These timing errors will tend to add up in the "wrong direction" as more devices are chained between MIDI Thru and MIDI In jacks. The result is that, regardless of circuit quality, there is a limit to the number of devices which can be chained (series-connected) in this fashion.
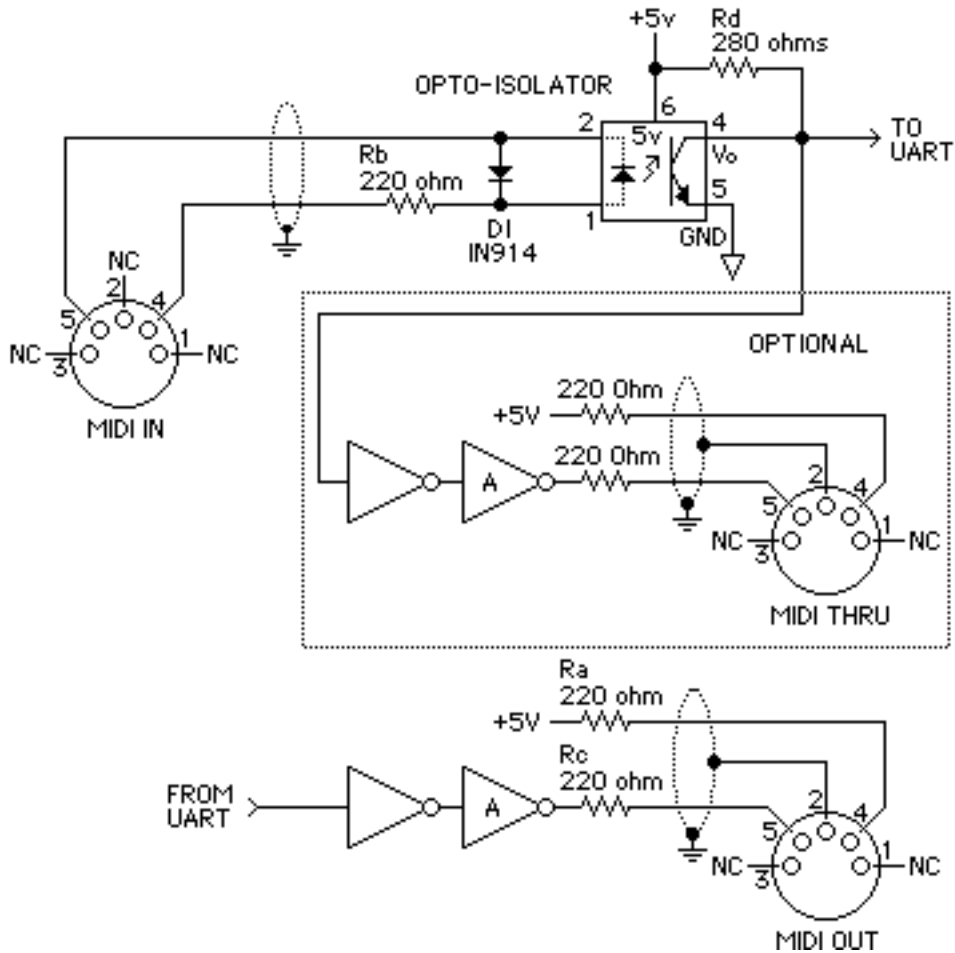
MIDI Standard Hardware

NOTES:

1. Opto-isolator currently shown is Sharp PC-900
   (HP 6N138 or other opto-isolator can be used with appropriate changes.)

2. Gates "A" are IC or transistor.

3. Resistors are 5%

Cables shall have a maximum length of fifty feet (15 meters), and shall be terminated on each end by a corresponding 5-pin DIN male plug, such as the SWITCHCRAFT 05GM5M. The cable shall be shielded twisted pair, with the shield connected to pin 2 at both ends.

A MIDI Thru output may be provided if needed, which provides a direct copy of data coming in MIDI In. For long chain lengths (more than three instruments), higher-speed opto-isolators should help to avoid additive rise/fall time errors which affect pulse width duty cycle.

DATA FORMAT

MIDI communication is achieved through multi-byte "messages" consisting of one Status byte followed by one or two Data bytes. Real-Time and Exclusive messages are exception.

A MIDI-equipped instrument typically contains a receiver and a transmitter. Some instruments may contain only a receiver or only a transmitter. A receiver accepts messages in MIDI format and executes MIDI commands. It consists of an opto-isolator, Universal Asynchronous Receiver/Transmitter (UART), and any other hardware needed to perform the intended functions. A transmitter originates messages in MIDI format, and transmits them by way of a UART and line driver.

MIDI makes it possible for a user of MIDI-compatible equipment to expand the number of instruments in a music system and to change system configurations to meet changing requirements.

MIDI messages are sent over any of 16 channels which are used for a variety of performance information. There are five major types of MIDI messages: Channel Voice, Channel Mode, System Common, System Real-Time and System Exclusive.

A MIDI event is transmitted as a "message" and consists of one or more bytes. The diagrams below show the structure and classification of MIDI data.

TYPES OF MIDI BYTES:

```
                        ┌──────────┐
                        │   Byte   │
                        └────┬─────┘
              ┌──────────────┴──────────────┐
      ┌───────────────┐            ┌───────────────┐
      │ Status Byte   │            │  Data Byte    │
      │ (80H - FFH)   │            │ (00H - 7FH)   │
      └───────────────┘            └───────────────┘
```

TYPES OF MIDI MESSAGES:

```
                        ┌──────────────┐
                        │ Message Type │
                        └──────┬───────┘
            ┌──────────────────┴──────────────────┐
      ┌──────────┐                          ┌──────────┐
      │ Channel  │                          │ System   │
      │ Message  │                          │ Message  │
      └────┬─────┘                          └────┬─────┘
      ┌────┴────┐              ┌──────────────────┼──────────────────┐
┌─────────┐┌─────────┐   ┌─────────┐       ┌─────────┐        ┌─────────┐
│ Channel ││ Channel │   │ System  │       │ System  │        │ System  │
│ Voice   ││ Mode    │   │Exclusive│       │ Common  │        │Real Time│
│ Message ││ Message │   │ Message │       │ Message │        │ Message │
└─────────┘└─────────┘   └─────────┘       └─────────┘        └─────────┘
```
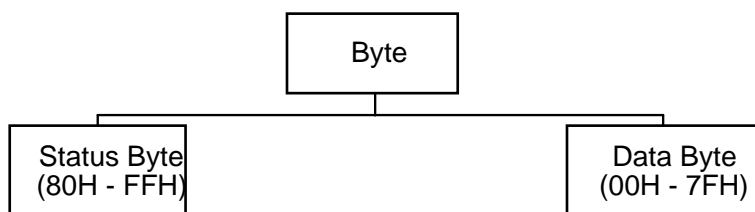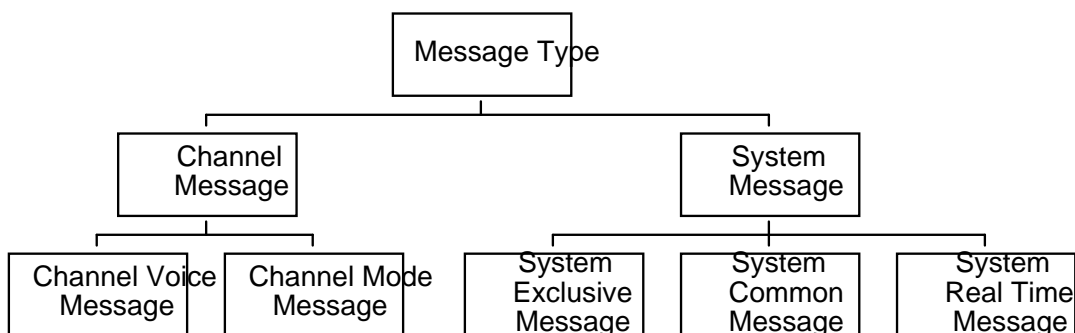
STRUCTURE OF A SINGLE MESSAGE:

Status

| Status | — | Data Byte |

| Status | — | Data Byte | — | Data Byte |
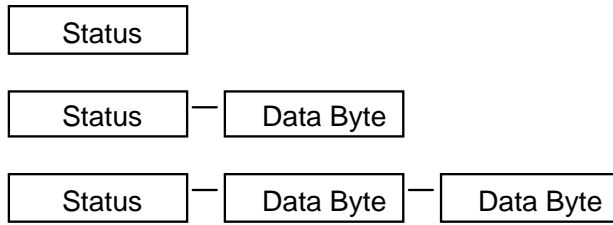
STRUCTURE OF SYSTEM EXCLUSIVE MESSAGES:

| Status | — | Data Bytes | — | EOX |

MESSAGE TYPES

Messages are divided into two main categories: *Channel* and *System*.

CHANNEL MESSAGES

A Channel message uses four bits in the Status byte to address the message to one of sixteen MIDI channels and four bits to define the message (see Table II). Channel messages are thereby intended for the receivers in a system whose channel number matches the channel number encoded into the Status byte.

An instrument can receive MIDI messages on more than one channel. The channel in which it receives its main instructions, such as which program number to be on and what mode to be in, is referred to as its "Basic Channel". An instrument may be set up to receive performance data on multiple channels (including the Basic Channel). These are referred to as "Voice Channels". These multiple-channel situations will be discussed in more detail later.

There are two types of Channel messages: *Voice* and *Mode*.

VOICE: To control an instrument's voices, Voice messages are sent over the Voice Channels.

MODE: To define the instrument's response to Voice messages, Mode messages are sent over an instrument's Basic Channel.

SYSTEM MESSAGES

System messages are not encoded with channel numbers. There are three types of System messages: *Common*, *Real-Time*, and *Exclusive*.

COMMON: Common messages are intended for all receivers in a system regardless of
             channel.

REAL-TIME: Real-Time messages are used for synchronization and are intended for all clock-
             based units in a system. They contain Status bytes only — no Data bytes. Real-
             Time messages may be sent at any time — even between bytes of a message
             which has a different status. In such cases the Real-Time message is either acted
             upon or ignored, after which the receiving process resumes under the previous
             status.

EXCLUSIVE: Exclusive messages can contain any number of Data bytes, and can be
             terminated either by an End of Exclusive (EOX) or any other Status byte (except
             Real Time messages). An EOX should always be sent at the end of a System
             Exclusive message. These messages include a Manufacturer's Identification (ID)
             code. If a receiver does not recognize the ID code, it should ignore the following
             data.

             So that other users and third party developers can fully access their instruments,
             manufacturers must publish the format of the System Exclusive data following
             their ID code. Only the manufacturer can define or update the format following
             their ID.

DATA TYPES

There are two types of bytes sent over MIDI: *Status Bytes* and *Data bytes*.

STATUS BYTES

Status bytes are eight-bit binary numbers in which the Most Significant Bit (MSB) is set (binary 1).
Status bytes serve to identify the message type, that is, the purpose of the Data bytes which follow it.
Except for Real-Time messages, new Status bytes will always command a receiver to adopt a new status,
even if the last message was not completed.

RUNNING STATUS

For Voice and Mode messages only. When a Status byte is received and processed, the receiver will
remain in that status until a different Status byte is received. Therefore, if the same Status byte would
be repeated, it can optionally be omitted so that only the Data bytes need to be sent. Thus, with Running
Status, a complete message can consist of only Data bytes.

Running Status is especially helpful when sending long strings of Note On/Off messages, where "Note
On with Velocity of 0" is used for Note Off.

Running Status will be stopped when any other Status byte intervenes. Real-Time messages should not
affect Running Status.

See also: Additional Explanations and Application Notes

UNIMPLEMENTED STATUS

Any status bytes, and subsequent data bytes, received for functions not implemented in a receiver should be ignored.

UNDEFINED STATUS

All MIDI devices should be careful to never send any undefined status bytes. If a device receives any such code, it should be ignored without causing any problems to the system. Care should also be taken during power-up and power-down that no messages be sent out the MIDI Out port. Such noise, if it appears on a MIDI line, could cause a data or framing error if the number of bits in the byte are incorrect.

DATA BYTES

Following a Status byte (except for Real-Time messages) there are either one or two Data bytes which carry the content of the message. Data bytes are eight-bit binary numbers in which the Most Significant Bit (MSB) is always set to binary 0. The number and range of Data bytes which must follow each Status byte are specified in the tables in section 2. For each Status byte the correct number of Data bytes must always be sent. Inside a receiver, action on the message should wait until all Data bytes required under the current status are received. Receivers should ignore Data bytes which have not been properly preceded by a valid Status byte (with the exception of "Running Status," explained above).

CHANNEL MODES

Synthesizers and other instruments contain sound generation elements called voices. Voice assignment is the algorithmic process of routing Note On/Off data from incoming MIDI messages to the voices so that notes are correctly sounded.

Note: When we refer to an "instrument" please note that one physical instrument may act as several virtual instruments (i.e. a synthesizer set to a 'split' mode operates like two individual instruments). Here, "instrument" refers to a virtual instrument and not necessarily one physical instrument.

Four Mode messages are available for defining the relationship between the sixteen MIDI channels and the instrument's voice assignment. The four modes are determined by the properties Omni (On/Off), Poly, and Mono. Poly and Mono are mutually exclusive, i.e., Poly disables Mono, and vice versa. Omni, when on, enables the receiver to receive Voice messages on all voice Channels. When Omni is off, the receiver will accept Voice messages from only selected Voice Channel(s). Mono, when on, restricts the assignment of Voices to just one voice per Voice Channel (Monophonic.) When Mono is off (Poly On), a number of voices may be allocated by the Receiver's normal voice assignment (Polyphonic) algorithm.

For a receiver assigned to Basic Channel "N," (1-16) the four possible modes arising from the two Mode messages are:

Mode	Omni		
1	On	Poly	Voice messages are received from all Voice channels and assigned to voices polyphonically.
2	On	Mono	Voice messages are received from all Voice Channels, and control only one voice, monophonically.
3	Off	Poly	Voice messages are received in Voice channel N only, and are assigned to voices polyphonically.
4	Off	Mono	Voice messages are received in Voice channels N through N+M-1, and assigned monophonically to voices 1 through M, respectively. The number of voices "M" is specified by the third byte of the Mono Mode Message.

Four modes are applied to transmitters (also assigned to Basic Channel N). Transmitters with no channel selection capability should transmit on Basic Channel 1 (N=1).

Mode	Omni		
1	On	Poly	All voice messages are transmitted in Channel N.
2	On	Mono	Voice messages for one voice are sent in Channel N.
3	Off	Poly	Voice messages for all voices are sent in Channel N.
4	Off	Mono	Voice messages for voices 1 through M are transmitted in Voice Channels N through N+M-1, respectively. (Single voice per channel).

A MIDI receiver or transmitter operates under only one Channel Mode at a time. If a mode is not implemented on the receiver, it should ignore the message (and any subsequent data bytes), or switch to an alternate mode, usually Mode 1 (Omni On/Poly).

Mode messages will be recognized by a receiver only when received in the instrument's Basic Channel — regardless of which mode the receiver is currently assigned to. Voice messages may be received in the Basic Channel and in other Voice Channels, according to the above specifications.

Since a single instrument may function as multiple "virtual" instruments, it can thus have more than one basic channel. Such an instrument behaves as though it is more than one receiver, and each receiver can be set to a different Basic Channel. Each of these receivers may also be set to a different mode, either by front panel controls or by Mode messages received over MIDI on each basic channel. Although not a true MIDI mode, instruments operating in this fashion are described as functioning in "Multi Mode."

An instrument's transmitter and receiver may be set to different modes. For example, an instrument may receive in Mono mode and transmit in Poly mode. It is also possible to transmit and receive on different channels. For example, an instrument may receive on Channel 1 and transmit on Channel 3.

POWER-UP DEFAULT CONDITIONS

It is recommended that at power-up, the basic channel should be set to 1, and the mode set to Omni On/Poly (Mode 1). This, and any other default conditions for the particular instrument, should be maintained indefinitely (even when powered down) until instrument panel controls are operated or MIDI data is received. However, the decision to implement the above, is left totally up to the designer.

CHANNEL VOICE MESSAGES

Note-Off	8nH
Note-On	9nH
Poly Key Pressure	AnH
Control Change	BnH (0 - 119)
Program Change	CnH
Channel Pressure	DnH
Pitch Bend	EnH

Channel Voice Messages are the bulk of information transmitted between MIDI instruments. They include all Note-On, Note-Off, program change, pitch-wheel change, after-touch pressure and controller changes. These terms are defined below.

A single Note-On message consists of 3 bytes, requiring 960 microseconds for transmission. When many notes are played at the same time, the multiple Note-On messages may take several milliseconds to transmit. This can make it difficult for MIDI to respond to a large number of simultaneous events without some slight audible delay. This problem can be relieved to some degree by using the Running Status mode described on page 5 and in the appendix (A1-3).

TYPES OF VOICE MESSAGES

NOTE-ON: Message is sent by pressing a key or from other triggering devices.

NOTE-OFF: Message is sent by releasing a key.

CONTROL CHANGE: Message is sent when a controller other than a key (e.g. a pedal, wheel, lever, switch, etc.) is moved in order to modify the sound of a note (e.g. introducing modulation, sustain, etc.). Control changes are not used for sending parameters of tones (voices), such as attack time, filter cut off frequency, etc.

PROGRAM CHANGE: When a "program" (i.e. sound, voice, tone, preset or patch) is changed, the number corresponding to the newly selected program is transmitted.

AFTER TOUCH: This message typically is sent by key after-pressure and is used to modify the note being played. After touch messages can be sent as Polyphonic Key Pressure or Channel Pressure.

PITCH BEND CHANGE: This message is used for altering pitch. The maximum resolution possible is 14 bits, or two data bytes.

Voice messages are not exclusively for use by keyboard instruments, and may be transmitted for a variety of musical purposes. For example, Note-On messages generated with a conventional keyboard synthesizer may be used to trigger a percussion synthesizer or lighting controller.

NOTE NUMBER

Each note is assigned a numeric value, which is transmitted with any Note-On/Off message. Middle C has a reference value of 60. This is the middle C of an 88 note piano-style keyboard though it need not be physically located in the center of a keyboard.

```
    0    12   24   36   48   60   72   84   96   108  120  127
    ─────────────────────────────────────────────────────────
              a c    c    c    c    c    c    c    c
              ├──────────────────────────────────┤
                           piano range
```
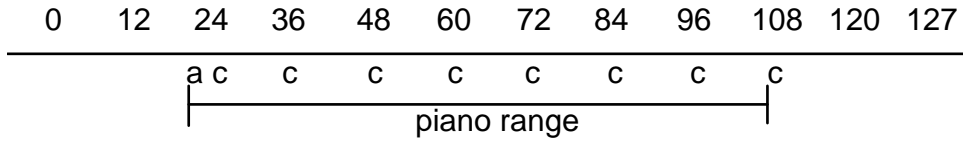
VELOCITY

Interpretation of the Velocity byte is left up to the receiving instrument. Generally, the larger the numeric value of the message, the stronger the velocity-controlled effect. If velocity is applied to volume (output level) for instance, then higher Velocity values will generate louder notes. A value of 64 (40H) would correspond to a mezzo-forte note and should also be used by device without velocity sensitivity. Preferably, application of velocity to volume should be an exponential function. This is the suggested default action; note that an instrument may have multiple tables for mapping MIDI velocity to internal velocity response.

```
    0      1                      64                 127
    ─────────────────────────────────────────────────────
    off   ppp    p     pp    mp    mf     f     ff   fff
```
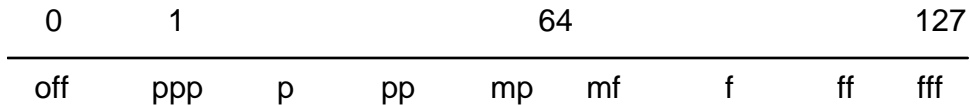
vvvvvvv = 64: if not velocity sensitive
vvvvvvv = 0: Note-Off (with velocity of 64)

NOTE-OFF

MIDI provides two roughly equivalent means of turning off a note (voice). A note may be turned off either by sending a Note-Off message for the same note number and channel, or by sending a Note-On message for that note and channel with a velocity value of zero. The advantage to using "Note-On at zero velocity" is that it can avoid sending additional status bytes when Running Status is employed.

Due to this efficiency, sending Note-On messages with velocity values of zero is the most commonly used method. However, some keyboard instruments implement release velocity where a Note-Off code (8nH) accompanied by a "velocity off" byte is used. A receiver must be capable of recognizing either method of turning off a note, and should treat them identically.

The three methods of using Note-On (9nH) or Note-Off (8nH) are as follows:

1. For a keyboard which does not implement Velocity, the note will be turned on using 9n, kkkkkkk, 64 (40H) and may be turned off using 9n, 0kkkkkkk, 00000000 or 8n, 0kkkkkkk, 0xxxxxxx (a value of 64 [40H] is used for x).

2. For a keyboard which incorporates Key On Velocity, but not Release Velocity the note is turned on using 9n 0kkkkkkk, 0vvvvvvv and may be turned off using 9n, 0kkkkkkk, 00000000 or 8n, 0kkkkkkk, 0xxxxxxx (a value of 64 (40H) is recommended for x).

3. Where the keyboard implements both Key On Velocity and Release Velocity, a note is turned on using 9n 0kkkkkkk, 0vvvvvvv, and turned off using 8n, 0kkkkkkk, 0vvvvvvv.

CONTROL CHANGE

The Control Change message is generally used for modifying tones with a controller other than a keyboard key. It is not for setting synthesizer parameters such as VCF cut-off, envelope decay, etc. There are some exceptions to the use of the Control Change message, such as the special Bank Select message and the RPN/NRPN messages (listed below).

CONTROLLER NUMBERS

All controller number assignments are designated by agreement between the MMA and JMSC. The numbers listed in Table III are specified for standard musical instrument applications. However, many non-musical devices which implement MIDI, such as lighting controllers, may use designated controller numbers at their discretion. Due to the limited number of controller numbers it would be impossible to assign a number to every possible effect (musical and non-musical) used now and in the future. For this reason, controllers are generally assigned only for purposes associated with musical instruments.

It is up to the manufacturer to inform their users of the fact that a device is using non-standard controller assignments. Though controllers may be used for non-musical applications, they must still be used in the format detailed in Table II. Manufacturers can request through the MMA or JMSC that logical controllers be assigned to physical ones as needed. A controller allocation table should be provided in the user's operation manual of all products.

A manufacturer wishing to control a number of device-specific parameters over MIDI should used non-registered parameter numbers and the Data Entry controllers (Data Entry Slider, Increment, and Decrement messages) as opposed to a large number of controllers. This alleviates possible conflict with devices responding to the same control numbers unpredictably.

There are currently 120 controller numbers, from 0 through 119 (controller 120 was recently adopted as a Channel Mode Message and is no-longer considered a Control Change). As shown below, controller numbers 32 to 63 are used to define an LSB byte for corresponding controllers 0 through 31. Controller classifications are as follows:

0	through	31	=	MSB of most continuous Controller Data
32	through	63	=	LSB for controllers 0 through 31
64	through	95	=	Additional single-byte controllers
96	through	101	=	Increment/Decrement and Parameter numbers
102	through	119	=	Undefined single-byte controllers

A numeric value (controller number) is assigned to the controllers of the transmitting instrument. A receiver may use the message associated with a controller number to perform any operation or achieve any desired effect. Further, a single controller number may be used to change a number of parameters.

controller numbers are classified by various categories. Each controller number corresponds to one byte of data.

Controller numbers 0 through 31 are for controllers that obtain information from pedals, levers, wheels, etc. Controller numbers 32 through 63 are reserved for optional use as the LSB (Least Significant Byte) when higher resolution is required and correspond to 0 through 31 respectively. For example, controller number 7 (Volume) can represent 128 steps or increments of some controller's position. If controller number 39, the corresponding LSB number to controller number 7, is also used, 14-bit resolution is obtained. This provides for resolution of 16,384 steps instead of 128.

If 128 steps of resolution is sufficient the second byte (LSB) of the data value can be omitted. If both the MSB and LSB are sent initially, a subsequent fine adjustment only requires the sending of the LSB. The MSB does not have to be retransmitted. If a subsequent major adjustment is necessary the MSB must be transmitted again. When an MSB is received, the receiver should set its concept of the LSB to zero.

All controller numbers 64 and above have single-byte values only, with no corresponding LSB. Of these, 64 through 69 have been defined for switched functions (hold pedal, etc.) while 91 through 95 are for controlling the depth of certain external audio effects.

Control numbers 64 through 69 are assigned to functions normally associated with switches (i.e. sustain or soft pedals). However these controllers can be used to send any continuous value. The reverse can also be true for a continuous controller such as Modulation Wheel. While this controller is most often used as a variable control, an on/off modulation switch can also be used. This would be accomplished by sending the Modulation Controller number (01) and a data byte of either 0 (off) or 127 (on).

If a receiver is expecting switch information it should recognize 0-63 (00H-3FH) as "OFF" and 64-127 (40H-7FH) as "ON". This is because a receiver has no way of knowing whether the message information is from a switch or a continuous controller. It is very important to always use an existing control number. The control numbers already adopted for use are listed in Table III. We will discuss some of them, but not all, below.

GLOBAL CONTROLLERS

If a receiving instrument is in Mode 4 (Omni Off/Mono) and is thus able to respond to more than one MIDI channel, it is possible to use a Global Controller to affect all voices regardless of MIDI channel. This is accomplished by sending any controller intended to affect all voices over the MIDI channel one below the basic channel of the receiver. For example, if a receiving synthesizer in Mode 4 is responding to channels 6 through 12, its basic channel is 6. Any controllers received on channel 5 would be Global Controllers and would affect all voices. If the Basic Channel is 1, then the Global Channel wraps to become 16, though not all receivers may provide this function.

GENERAL PURPOSE CONTROLLERS

Controller numbers 16-19 and 80-83 are defined as General Purpose Controllers. They may be used by a manufacturer for any added functions able to send or receive some sort of control information needed for a specific product. They do not have any intrinsic functions assigned to them. General Purpose Controllers 16-19 are two byte controllers (with controller numbers 48-51 for an optional LSB). General Purpose Controllers 80-83 are single byte controllers. As an example, an instrument with a special, user definable joystick or lever assignable to any internal parameter could send and receive General Purpose Controller numbers for sequencing.

CONTROLLER EFFECT

All transmitters should send a value of 00 to represent minimum and 127 (7FH) to represent maximum. For continuous controllers without a center detented position, it is recommended that the minimum effect position correspond to 00, and the maximum effect position correspond to 127 (7FH).

Virtually all controllers are defined as 0 being no effect and 127 being maximum effect. There are three defined controllers that are notably different: Balance, Pan and Expression.

> BALANCE: A Balance Controller has been adopted as continuous controller number 8 (08H) with value 00 = full volume for the left or lower half, 64 (40H) = equal balance, and 127 (7FH) = full volume for the right or upper half. This controller determines the volume balance between two different sound sources.
>
> PAN: A Pan Controller has been adopted as continuous controller number 10 (0AH) with value 00 = hard left, 64 (40H) = center, and 127 (7FH) = hard right. This controller determines where a single sound source will be located in a stereo field.
>
> EXPRESSION: An Expression Controller has been adopted as continuous controller number 11 (0BH). Expression is a form of volume accent above the programmed or main volume.

BANK SELECT

Bank Select is a special controller. The Bank Select message is an extension of the Program Change message and is used to switch between multiple banks. For example, a bank select message could be used to select more than 128 programs, or switch between internal memory and external RAM card.

Control Change numbers 00H and 20H are defined as the Bank Select message. 00H is the MSB and 20H is the LSB for a total of 14 bits. This allows 16,384 banks to be specified.

The transmitter must transmit the MSB and LSB as a pair, and the Program Change must be sent immediately after the Bank Select pair. If their is any delay between these messages and they are passed through a merging device (which may insert another message) the message may be interpreted incorrectly.

The messages Bank Select MSB, LSB and Program number will select a specific program. After switching to another bank, any Program Change messages transmitted singularly will select other program in that bank.

After the receiver as received the entire Bank Select messages it will normally change to a new program. The program must change upon the receipt of the Program Change message. However, the program need not be changed for a note which is already sounding. When the Bank Select message is received, the receiving device must remember that bank number in readiness for the following Program Change. Bank Select alone must not change the program. This is to assure that multiple devices change concurrently.

The 14 bit Bank Select value corresponds to bank numbers as follows:

MSB	LSB	Bank Number
00H	00H	Bank 1
00H	7FH	Bank 128
01H	00H	Bank 129
7FH	7FH	Bank 16,384

As with program numbers, banks begin counting from 1. Thus the actual bank number will be (MIDI value + 1).

LEGATO FOOTSWITCH

```
Bn 44 vv      Legato Footswitch
              vv = 00-3F   Normal
              vv = 40-7F   Legato
```

Legato Footswitch is a recent addition to the specification. This controller is used to turn monophonic legato response in a receiving instrument on and off. When turned on the instrument goes into a monophonic mode; if a new Note-On is received before the Note-Off for the currently sounding note, pitch is changed without re-attacking the envelopes or (if possible) playing the attack portion of the sound. When turned off the voice assignment mode (polyphonic or monophonic) returns to the state it was in prior to receiving the Legato On command.

Note: This message is not a replacement for proper Mode 4 legato operation. Nor is it a replacement for sending Note-Offs for every Note-On sent. It is specifically intended as a useful performance controller.

EFFECTS CONTROLLER REDEFINITION

Controller numbers 91 – 95 are defined as Effects Depth 1 through Effects Depth 5 and can be used for controlling various effects. Their former titles of External Effects Depth, Tremolo Depth, Chorus Depth, Celeste (Detune) Depth, and Phaser Depth are now the recommended defaults.

SOUND CONTROLLERS

Controllers 46H through 4FH are defined as "Sound Controllers." Manufacturers and users may map any functions they desire to these ten controllers. However, to further aid standardization and easy set-up for users, the MMA and JMSC may determine "default" assignments for these controllers. A manufacturer may independently assign other functions to these controllers, but it should be understood that the MMA and JMSC may later assign different defaults to them.

Five Sound Controller defaults have currently been defined by the MMA and JMSC:

Number	Name	Instruments
46H (70)	Sound Controller #1	Sound Variation
47H (71)	Sound Controller #2	Timber/Harmonic Intensity
48H (72)	Sound Controller #3	Release Time
49H (73)	Sound Controller #4	Attack Time
4AH (74)	Sound Controller #5	Brightness

SOUND VARIATION CONTROLLER:

        Bn 46 vv   Sound Variation

This controller is used to select alternate versions of a sound during performance. Note that it is different from a program change in several ways:

   1. The variation (alternate sound) is an intrinsic part of the program which is being played, and is programmed in the patch.

   2. The variation is usually related to the primary sound – for example, a sax and an overblown sax, bowed and pizzicato strings, a strummed and muted guitar, etc.

   3. The variation to be used is decided at the time of the Note-On. For example, if the value of SVC is set to 00, notes are sounded, and then SVC is changed to 24H, the notes currently sounding will not change. Any new notes will take the variation determined by the new SVC value. If the old notes are released, they will finish in their original manner.

SVC actually acts as a multi-level switch. An instrument's levels of variations should be mapped over the entire 00-7FH range of the controller. For example, if an instrument had only a single SVC switch, it would transmit a value of 00) for the primary sound and 7FH for the secondary sound. If an instrument had four variations, it would transmit these as 00, 20H, 40H, and 7FH. The first instrument would receive any value in the range of 40H-7FH to select its secondary sound.

TIMBRE CONTROLLERS:

        Bn 47 vv   Timbre/Harmonic Intensity
        Bn 4A vv   Brightness

The Harmonic Content controller (commonly known as "timbre") is intended as a modifier of the harmonic content of a sound, e.g. FM feedback, FM modulation amount, waveform loop points, etc. The receiving instrument determines the application of this controller according to its voice architecture.

Harmonic Content should be treated as an absolute rather than relative value, and should be handled as a modulation input by the receiver.

The Brightness controller is aimed specifically at altering the brightness of the sound. In most sound modules, it would correspond to a low pass filter's cutoff frequency, although it might also control EQ or a harmonic enhancer/exciter.

Brightness should be treated as a relative controller, with a data value of 40H meaning no change, values less than 40H meaning progressively less bright, and values greater than 40H meaning progressively more bright.

Both of these controllers are intended as a performance controllers – not as a sound parameter editing controllers (in other words, these messages do not change the memorized data of a preset). They have no fixed association with any physical controller.

The receiving instrument should be able to respond to these controllers while sustaining notes without audible glitches or re-triggering of the sound. The effective range of these controllers may be programmed per preset, if desired.

ENVELOPE TIME CONTROLLERS:

```
Bn 48 vv      Release Time
Bn 49 vv      Attack Time
              vv = 00 - 3F  = shorter times (00 = shortest adjustment)
              vv = 40       = no change in times
              vv = 41 - 7F  = longer times (7F = longest adjustment)
```

These controllers are intended to adjust the attack and release times of a sound relative to its pre-programmed values. The manufacturer and user should decide which envelopes in a voice are affected; the default should be all envelopes. These controllers should affect all envelopes affected about to enter their release or attack phases (respectively); the manufacturer may allow an option to affect envelope phases already started.

These envelope time controllers do not replace the effect attack or release velocity may have on the envelope times of the sound; they should interact with them in a predictable manner. They have no fixed association with a physical controller. The effective range of these controllers may be programmed per preset, if desired.

These are intended as a performance controllers – not as a sound parameter editing controllers (in other words, these messages do not change the memorized data of a preset). The receiving instrument should be able to respond to these controllers while sustaining notes and without audible glitches or re-triggering of the sound.

PORTAMENTO CONTROLLER

```
Bn 54 kk
              n       = channel
              kk      = source note number for pitch reference
```

Portamento Control (PTC) is a recent addition, and defines a continuous controller that communicates which note number the subsequent note is gliding from. It is intended for special effects in playing back pre-sequenced material, so that legato with portamento may be realized while in Poly mode.

When a Note-On is received after a Portamento Control message, the voice's pitch will glide from the key specified in the Portamento Control message to the new Note-On's pitch at the rate set by the portamento time controller (ignoring portamento on/off).

A single Portamento Control message only affects the next Note-On received on the matching channel (i.e. it is reset after the Note-On). Receiving a Portamento Control message does not affect the pitch of any currently sustaining or releasing notes in Poly modes; if in Mono mode or if Legato Footswitch is currently on, a new overlapped note event would result in an immediate pitch jump to the key specified by the portamento Control message, and then a glide at the current portamento rate to the key specified by the new Note-On.

In all modes, the note is turned off by a note that matches the Note-On key number; not the key number stated in the Portamento Control message. Pitch bend offsets the pitch used by both the Portamento Control starting note and the target Note-On.

If there is a currently sounding voice whose note number is coincident with the source note number, the voice's pitch will glide to the new Note-On's pitch according to the portamento time without re-attacking. Then, no new voice should be assigned.

The single Portamento Control message only affects the next Note-On received on the matching channel (in other words, it is reset after the Note-On). Receiving a Portamento Control message does not affect the pitch of other currently sounding voices except a voice whose note number is coincident with the source key number of the Portamento Control message in Poly mode.

Example 1:

MIDI Message	Description	Result
`90 3C 40`	Note-On #60	#60 on (middle C)
`B0 54 3C`	PTC from #60	no change in current note
`90 40 40`	Note-On #64	re-tune from #60 to #64
`80 3C 40`	Note-Off #60	no change
`80 40 40`	Note-Off #64	#64 off

Example 2:

MIDI Message	Description	Result
`B0 54 3C`	PTC from #60	no change
`90 40 40`	Note-On #64	#64 with glide from #60
`80 40 40`	Note-Off #64	#64 Off

REGISTERED AND NON-REGISTERED PARAMETER NUMBERS

Registered and Non-Registered Parameter Numbers are used to represent sound or performance parameters. As noted below, Registered Parameters Numbers are agreed upon by the MMA and JMSC. Non-Registered Parameter Numbers may be assigned as needed by individual manufacturers. The basic procedure for altering a parameter value is to first send the Registered or Non-Registered Parameter Number corresponding to the parameter to be modified, followed by the Data Entry, Data Increment, or Data Decrement value to be applied to the parameter.

There are several rules and suggestions as to the use of these parameter numbers and controllers:

1. A manufacturer may assign any desired parameter to any Non-Registered Parameter Number. This list should be published in the owner's manual.

2. Reception of Non-Registered Parameter Numbers should be disabled on power-up to avoid confusion between different machines. Transmission of these numbers should be safe at any time if this is done.

3. After the reception of Non-Registered (or Registered) Parameter Numbers has been enabled, the receiver should wait until it receives both the LSB and MSB for a parameter number to ensure that it is operating on the correct parameter.

4. The receiver should be able to respond accordingly if the transmitter sends only an LSB or MSB to change the parameter number. However, since the transmitter can't know when reception was enabled on the receiver which will be waiting for both the LSB and MSB (at least initially), it is recommended that the LSB and MSB be sent each time a new parameter number is selected.

5. The Registered Parameter Numbers are agreed upon by the MMA and JMSC. Since this is a standardized list, reception of these Registered Parameter Numbers may be enabled on power-up.

6. Once a new Parameter Number is chosen, that parameter retains its old value until a new Data Entry, Data Increment, or Data Decrement is received.

PITCH BEND SENSITIVITY:

Pitch Bend Sensitivity is defined as Registered Parameter Number 00 00. The MSB of Data Entry represents the sensitivity in semitones and the LSB of Data Entry represents the sensitivity in cents. For example, a value of MSB=01, LSB=00 means +/- one semitone (a total range of two semitones).

MASTER TUNING:

Registered Parameter numbers 01 and 02 are used for Master Tuning control. They are implemented as follows:

RPN 01 - FINE TUNING:

Resolution:	100/8192 cents
Range:	100/8192* (-8192) to 100/8192* (+8191)

Control	Value	Displacement in cents from A440
MSB	LSB	
00	00	100/8192* (-8192)
40H	00	100/8192* (0)
7FH	7FH	100/8192* (+8191)

RPN 02 - COARSE TUNING:

Resolution:	100 cents
Range:	100* (-64) to 100* (+63)

Control	Value	Displacement in cents from A440
MSB	LSB	
00	XX	100* (-64)
40H	XX	100* (0)
7FH	XX	100* (+63)

PROGRAM CHANGE

This message is used to transmit the program or "patch" number when changing sounds on a MIDI instrument. The message does not include any information about the sound parameters of the selected tone. As the various parameters that constitute a program are very different from one MIDI instrument to another it is much more efficient to address a sound simply by its internal number.

Program Change messages are most often sent when physically selecting a new sound on an instrument. However, if the transmitting instrument does not produce its own sound, a button or any other physical controller can be used for transmitting program change messages to slave devices.

It is not often that the exact same tones are in the transmitting and receiving instruments, so some care must be taken when assigning tones to a given tone number. The ability to reassign programs to a given program change number should be part of an instrument's capabilities. Some instruments number their internal patches in octal numerics. This should have no effect on the numbers used for patch change. Numbering should begin with 00H and increment sequentially. For example, octal 11 would be 00H, 12 would be 01H, etc.

It may not always be desirable for a tone change in a transmitting instrument to cause a program change in a receiving instrument. Some means of disabling the sending or reception of program change should be provided. Program change messages do not necessarily need to change tones. In some instruments, such as a drum machine, the message may be used to switch to a different rhythmic pattern. In MIDI controlled effects devices, the program change message may be used to select a different preset effect.

Note: also see Bank Select.

PITCH BEND CHANGE

This function is a special purpose pitch change controller, and messages are always sent with 14 bit resolution (2 bytes). In contrast to other MIDI functions, which may send either the LSB or MSB, the Pitch Bender message is always transmitted with both data bytes. This takes into account human hearing which is particularly sensitive to pitch changes. The Pitch Bend Change message consists of 3 bytes when the leading status byte is also transmitted. The maximum negative swing is achieved with data byte values of 00, 00. The center (no effect) position is achieved with data byte values of 00, 64 (00H, 40H). The maximum positive swing is achieved with data byte values of 127, 127 (7FH, 7FH).

Sensitivity of Pitch Bend Change is selected in the receiver. It can also be set by the receiver or transmitted via Registered Parameter number 00 00.

AFTERTOUCH

Two types of Aftertouch messages are available: one that affects an entire MIDI channel and one that affects each individual note played. They are differentiated by their status byte. In either case, the Aftertouch value is determined by horizontally moving the key (front-to-rear or left-to-right), or by pressing down on the key after it "bottoms out". Devices such as wind controllers can send Aftertouch from increasing breath pressure after the initial attack. The type of tone modification created by the Aftertouch is determined by the receiver. Aftertouch may be assigned to affect volume, timbre, vibrato, etc.

If a "Channel Pressure" (Dn, 0vvvvvvv) message is sent, then the Aftertouch will affect all notes playing in that channel.

If a "Polyphonic Key Pressure" (An, 0kkkkkkk, 0vvvvvvv) message is sent discrete Aftertouch is applied to each note (0kkkkkkk) individually.

CHANNEL MODE MESSAGES

(Control Change Status)	BnH
All Sound Off	120
Reset All Controllers	121
Local Control	122
All Notes Off	123
Omni Off	124
Omni On	125
Mono On (Poly Off)	126
Poly On (Mono Off)	127

A Mode message is sent with the same Status Byte as a Control Change message. The second byte of the message will be between 121 (79H) and 127 (7FH) to signify a mode message. Mode messages determine how an instrument will receive all subsequent voice messages. This includes whether the receiver will play notes monophonically or polyphonically and whether it will respond only to data sent on one specific voice channel or all of them.

MODE MESSAGES AS ALL NOTES OFF MESSAGES

Messages 123 through 127 also function as All Notes Off messages. They will turn off all voices controlled by the assigned Basic Channel. These messages should not be sent periodically, but only for a specific purpose. In no case should they be used in lieu of Note-Off commands to turn off notes which have been previously turned on. Any All Notes Off command (123-127) may be ignored by a receiver with no possibility of notes staying on, since any Note-On command must have a corresponding specific Note-Off command.

THE BASIC CHANNEL OF AN INSTRUMENT

Mode messages are recognized only when sent on the Basic Channel to which the receiver is assigned, regardless of the current mode. The Basic Channel is set in the transmitter or receiver either by permanent "hard wiring," by panel controls, or by System Exclusive messages, and cannot be changed by any MIDI mode or voice message. Mode messages can only be transmitted and received on an instrument's Basic Channel.

RECEIVER'S MODE (OMNI ON/OFF & POLY/MONO)

The receiver can be set to any of four modes which determine how it will recognize voice messages. The four modes are set with two mode messages: Omni On/Off, and Poly/Mono.

Mode 1:	Omni On, Poly
Mode 2:	Omni On, Mono
Mode 3:	Omni Off, Poly
Mode 4:	Omni Off, Mono

Mono and Poly determine how the receiver's voices will be assigned when more than one note is received simultaneously. In Mono mode, each voice in the receiver will respond monophonically to note messages on a particular MIDI channel. This would be like having several Monophonic synthesizers in a single box. In Poly mode, voices in the receiver will respond to note messages polyphonically.

These four modes may be changed by panel controls on the receiver. Care must be taken, however, since it is possible that the receiver may be "disabled" by setting it to a mode where it will not recognize or correctly respond to data received from a transmitter. As the receiver has no way of knowing the mode of the transmitter, there is no guarantee that a receiver will interpret messages as expected if it has been manually set to a different mode.

The recommended start up condition is Omni Mode On. This allows two instruments to be connected and played immediately without concern for selecting the instruments' basic channel. The receiver will respond to voice messages on all MIDI channels. With Omni off, a receiver would only respond to the voice messages on the Basic Channel to which it is set.

Voice Message Paths with Poly/Mono and Omni On/Off Mode Selections:
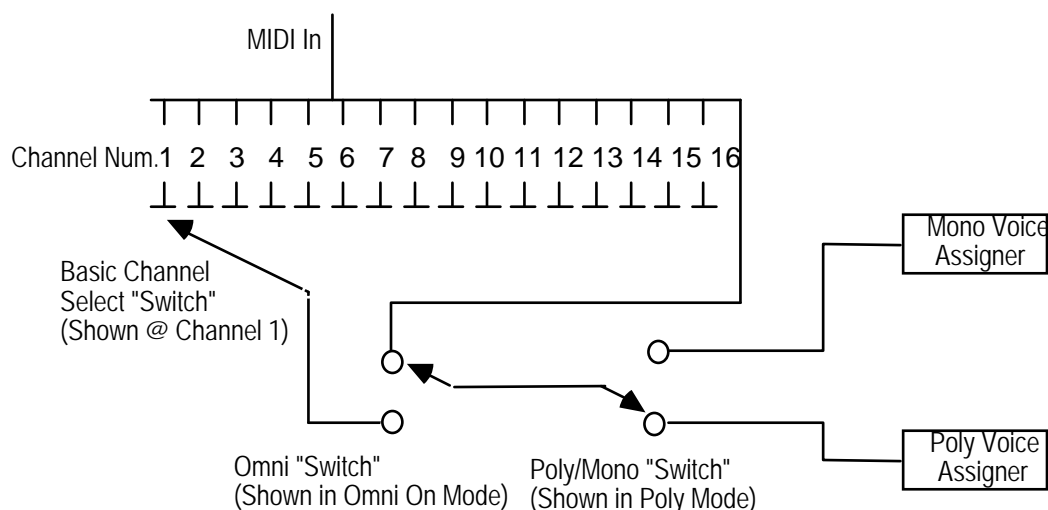
When the receiver is in Poly mode and more than one note is received on the recognized channel(s), those notes will be played simultaneously to the limit of the receiver's number of voices. The recognizable channel(s) refers to all MIDI channels when Omni is On, or to only the receiver's Basic Channel when Omni is Off.

When the receiver is in Mono mode, notes are assigned differently depending on whether Omni mode is On or Off.

MONO MODE

Mono mode is particularly useful for receiving MIDI data from guitar controllers, but can be used with keyboards and other controller devices as well. It is useful for such purposes as independent pitch bending of individual notes, portamento at specific rates between two notes, or transposition effects.

One of the reasons to use Mono mode is so that a receiver may respond in legato fashion to incoming note messages. If a Note-On is received, and then a second Note-On received without the first Note-Off being received, then the receiving instrument should change pitch to the new note, but not restart the envelopes (they should continue as if the same note was still being held). For a transmitter wishing a receiver to respond in legato fashion, the timing of the note messages would be like this:

<Note-On #1> <Note-On #2> <Note-Off #1>, etc.

MIDI rules still apply - a Note-Off must eventually be sent for every note. *Also see Legato Mode.*

OMNI-OFF/MONO

The third byte of a Mono Message specifies the number of channels in which Monophonic Voice messages are to be sent. If the letter M represents the number of acceptable MIDI channels, which may range from 1 to 16, and the letter N represents the basic channel number (1-16), then the channel(s) being used will be the current Basic Channel (N) through N+M-1 up to a maximum of 16. M=0 is a special case directing the receiver to assign all its voices, one per channel, from the Basic Channel N through 16.

TRANSMITTER:	When a transmitter set to Omni-Off/Mono mode, voice messages are sent on channels N through N+M-1. This means that each individual voice (or note) is sent on a single channel. The number of transmitted channels is limited to the number of voices in the transmitter. Additional notes will be ignored. When transmitting from a 16 voice instrument whose basic channel number N is set higher than 1, N+M-1 will be greater than 16 and notes assigned to nonexistent channels above 16 should not be sent. If full 16 voice transmission is possible, the basic channel N should be set to 1. For example, a four-voice instrument set to a basic channel of 3 would transmit note messages on channels 3, 4, 5 and 6.
RECEIVER:	In a receiver set to Omni-Off/Mono mode, the voice messages received in channels N through N+M-1 are assigned monophonically to its internal voices 1 through M. If N=1, and M=16 (maximum), then the messages are received on Channels 1 through 16. Should more than one Note-On message be sent for a given channel, the receivers response is not specified. Only one note (or voice) can be assigned to a given MIDI channel in this mode. M=0 is a special case directing the receiver to assign its voices, one per channel, from the basic channel N through 16, until all available voices are used.

OMNI-ON/MONO

When a transmitter is set to Omni-On/Mono mode, voice messages for a single voice are sent on channels N. If a receiver is set to Omni-On/Mono mode, then voice messages received from any voice channel will control a single voice monophonically. Regardless of the number of MIDI channels being received or the polyphony on any of them, the receiver will only play one note at a time.

TRANSMITTER:	A transmitter may send a Mono message to put a receiver into Mono mode. However, since a receiver may not be capable of Mono mode, the transmitter may continue to send note messages polyphonically. Even if the transmitter and receiver are both playing monophonically, multiple Note-On messages can be sent .

RECEIVER: When a Note-On message is sent in the Omni-On/Mono mode, the receiver will play that note regardless of channel number. If the value of M is 2 or greater when receiving a Mono On message and Omni is on, M is ignored and the receiver will still be monophonic. When Omni is on, it is inappropriate to send a Mono message with M greater than 1.

If a particular channel mode is not available on a receiver, it may ignore the message, or it may switch to an alternate mode (usually mode 1, which is Omni On/Poly).

MODES NOT IMPLEMENTED IN A RECEIVER

A transmitter could possibly request a mode not implemented in a receiver. For example, a transmitter might request Omni-Off Mono with M=2, but the receiver has only Omni-On Mono or Omni-On Poly capability. In this situation the receiver could do one of two things: (1) It can ignore the request, and no notes will sound; or (2) it can change to Omni-On Poly, and notes from both channels will play. The latter choice is recommended so that the receiver will respond to notes from the different channels.

	MONO		POLY
	M=1	M 1	
Omni Off	use Mode 2	use Mode 1	Mode 3
Omni On	Mode 2		Mode 1

If Receiver does not have Mode 4, it can disregard messages, or act as shown here

There is no way for a transmitter to know if a receiver has responded correctly to a particular Mode message. By implementing the response outlined above, unexpected results will be minimized. If it is possible for a receiver to ignore continuous controllers, it should do so in order that pitch bend or modulation intended for a single voice will not affect all the voices in the receiver.

A transmitter may send Omni On or Omni Off messages and Poly or Mono messages in any order. A receiver should set individual flags indicating Omni On/Off and Poly/Mono.

A receiver unable to accommodate a mode message such as Omni-Off Mono may switch to an alternate mode such as Omni-On Poly. If an Omni-Off message is then received, the receiver should not change to Omni-Off unless a Poly message is also received.

It is acceptable to repeat either of the Omni On/Off or Poly/Mono messages when changing modes. For example, if a transmitter sends Omni Off Poly and later sends Omni On Poly, the retransmission of the "Poly" message should cause no problem. If a receiver cannot accommodate an Omni-Off Mono mode change from a transmitter, it should switch to an alternate mode such as Omni-On Poly as outlined above.

Receive Mode Messages

Omni Off   Mono   Omni Off   Poly

Omni Flag   On = 1   Off = 0 | 1 | | 0 | | 0 |

Mono/Poly   Mono = 1   Flag   Poly = 0 | 0 | | 1 | | 0 |

(Power-Up Status)

Receiver Switches to Omni-Off

Unit Changes to Omni-On/Poly (Mode 1)

Receiver Changes to Omni-Off/Poly (Mode 3)
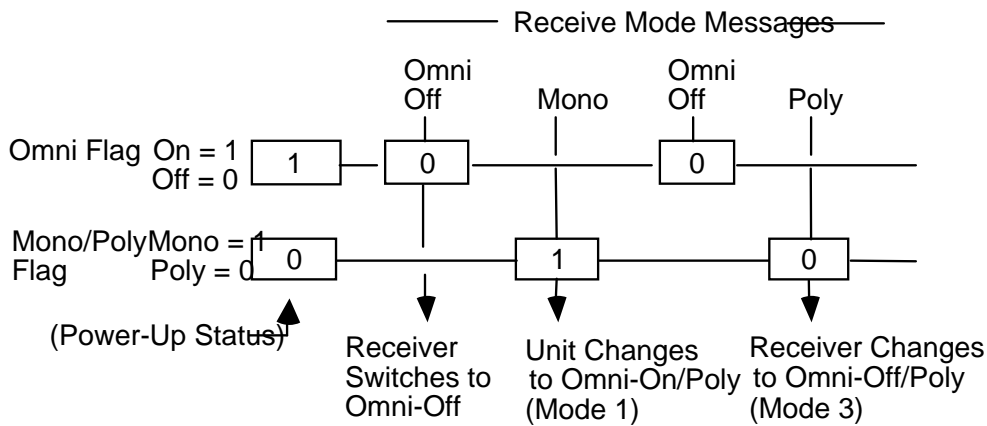
Receive Mode Messages

	Power Up	Omni-Off	Mono M >= 2	Omni-Off	Poly
OMNI	1	0	0	0	0
MONO/POLY	0	0	1	1	0
Possible Receive Modes	1	3	N/A	N/A	3
Alternate Receive Modes			1 (Omni-On, Poly)	1 (No Change)	

OMNI-ON = 1, OMNI-OFF = 0   N/A = A Mode Which is not Implemented in the Rec
MONO = 1, POLY = 0

ALL NOTES OFF

All Notes Off (123) is a mode message which provides an efficient method of turning off all voices turned on via MIDI. While this message is useful for some applications, there is no requirement that a receiver recognize it. Since recognition of All Notes Off is not required of the receiver, all notes should first be turned off by transmitting individual Note-Off messages prior to sending an All Notes Off.

In a MIDI keyboard instrument, notes turned on via the local keyboard should be differentiated from notes turned on via MIDI In.

MIDI Out   MIDI In   MIDI Thru   (Optional)

Press a Key   VOICE
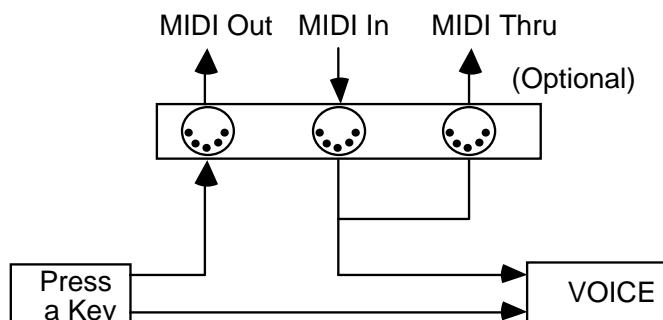
In an instrument structured as shown above, it is possible that the instrument may not differentiate between MIDI In and the local keyboard commands. If an All Notes Off is received via MIDI, then *all* notes will be turned off, including those being played on the instrument's own keyboard. This is not correct implementation of the All Notes Off message. All Notes Off should only turn off those notes that were turned on via MIDI. If an instrument cannot differentiate between its local keyboard and incoming MIDI messages, All Notes Off should be ignored.

Receivers should ignore an All Notes Off message while Omni is on (Modes 1 & 2). For example, if a receiver gets an All Notes Off from a sequencer that inserts such messages whenever all keys are released on a track, and two tracks were recorded on such a sequencer (even on different MIDI channels), the All Notes Off message would cut off sustaining notes recorded on the other track.

While MIDI devices should be able to respond to the All Notes Off message, an All Notes Off message should not be sent periodically as part of normal operation. This message should only be used to indicate that the entire MIDI system is "at rest" (i.e. when a sequence has stopped). However, a receiver should respond to an All Notes Off (unless Omni is on) whenever it is received, even when the system is not "at rest".

Although other mode messages will turn off all notes, they should not be used as a substitute for the All Notes Off message when desiring to turn off all notes. When the receiver is set to Omni-Off Poly mode (Mode 3), All Notes Off will cancel Note-On messages on the basic channel only. When a receiver is set to Omni-Off Mono mode (Mode 4), All Notes Off should only cancel Note-On messages on the channel over which the message was received.

Note: See more on All Notes Off in the Additional Explanations and Application Notes.

ALL SOUND OFF

All Sound Off (120) is a mode message intended to silence all notes currently sounding by instruments receiving on a specific MIDI channel. Upon reception, all notes currently on are turned off and their volume envelopes are set to zero as soon as possible.

This message is not a replacement for the All Notes Off message, Note-Off messages, Hold Off, or Master Volume Off. The correct procedure of sending a Note-Off for each and every Note-On must still be followed.

Although originally intended for silencing notes on a MIDI sound module, the All Sound Off message may be used to turn off all lights at a MIDI-controlled lighting console or to silence and clear the audio buffer of a MIDI-controlled reverb of digital delay.

RESET ALL CONTROLLERS

When a device receives the Reset All Controllers message (121), it should reset the condition of all its controllers (continuous and switch controllers, pitch bend, and pressures) to what it considers an ideal initial state (Mod wheel to 0, Pitch Bend to center, etc.). Reception follows the same rules as All Notes Off — Ignore if Omni is On .

Sequencers that wish to implement Reset All Controllers, but want to accommodate devices that do not implement this command, should send what they believe to be the initial state of all controllers first, followed by this message. Devices that respond to this message will end up in their preferred state, while those that do not will still be in the sequencer's chosen initialized state.

LOCAL CONTROL

Channel Mode Message 122, Local Control, is used to interrupt the internal control path between the keyboard and the sound-generating circuitry of a MIDI synthesizer. If 0 (Local Off) is received the path is disconnected, keyboard data goes only to MIDI Out and the sound-generating circuitry is controlled only by incoming MIDI data. If a 7FH (Local On) is received, normal operation is restored. Local Control should be switchable from an instrument's front panel.

When a keyboard instrument is being used as a slave device via MIDI, it may be desirable to disconnect the instrument's keyboard from its internal synthesizer so that local performance cannot interfere with incoming data. This may also save scanning time and thus speed up response to MIDI information. Instruments should power-up in Local On mode. An instrument should continue to send MIDI information from its keyboard while in Local Off.

SYSTEM COMMON MESSAGES

MIDI Time Code Quarter Frame	F1H
Song Position Pointer	F2H
Song Select	F3H
Tune Request	F6H
EOX (End of Exclusive)	F7H

MTC QUARTER FRAME

For device synchronization, MIDI Time Code uses two basic types of messages, described as Quarter Frame and Full. There is also a third, optional message for encoding SMPTE user bits. The Quarter Frame message communicates the Frame, Seconds, Minutes and Hours Count in an 8-message sequence. There is also an MTC FULL FRAME message which is a MIDI System Exclusive Message.

See the separate MTC specification document for complete details.

SONG POSITION POINTER

A sequencer's Song Position (SP) is the number of MIDI beats (1 beat = 6 MIDI clocks) that have elapsed from the start of the song and is used to begin playback of a sequence from a position other than the beginning of the song. It is normally set to 0 when the START button is pressed to start sequence playback from the very beginning. It is incremented every sixth MIDI clock until STOP is pressed. If CONTINUE is pressed, it continues to increment from its current value. The current Song Position can be communicated via the Song Position Pointer message and can be changed in a receiver by an incoming Song Position Pointer message. This message should only be recognized if the receiver is set to MIDI sync (external) mode.

Song Position Pointer is always multiplied by 6 times the MIDI clocks (F8H). Thus the smallest Song Position change is 6 MIDI clocks, or 1/16 note. The result is then multiplied by the internal time base of the sequencer. Here is an example:

> If Song Position Pointer = 10
>
> Multiply this times 6 MIDI clocks (10 X 6 = 60)
>
> Multiply the result (60) times the sequencer time base. If the time base is 96 clocks per beat, there are four internal clocks between each F8 so the result is 240 (60 X 4 = 240)
>
> Set internal pointers to begin playback 240 clock tics into the sequence.

Push
'Start'
Button

Internal Computation:
10(sp) x 6(clk) x 4(int) = 240

Wait for 'continue' Start
command at the position of
240 internal clocks

Push
'Stop'
Button

Push
'Continue'
Button

Enter Command:
'Song Position Pointer = 10'

FA   **FC**   **FB**

MIDI Clocks

Transmitter
Internal
Sequence

Ta   F8   F8   . . .   F8   F8   F8   F8   F8   F8

     1 2 3 4   56 57 58 59 60 61 62 63 64   240   242   244   246   248

Internal Clocks   Stop Play

MIDI Sync

Receive   Receive   Receive   Receive
FA   FC   SPP = 10   FB

MIDI Clocks

Receiver
Internal
Sequence

t   F8   F8   . . .   F8   F8   F8   F8   F8   F8

     1 2 3 4   56 57 58 59 60 61 62 63 64   240   242   244   246   248

Internal Clocks

Start Synched Sequence   Stop Sequence   Start by
                                                                     'Continue'
                                                                     Command

Count up Internal Clock to
240 clock position and wait
to receive 'FB' Command.

NOTES:
1. This example assumes an internal clock rate of 96 internal
   clocks per quarter note.

2. The time delay intervals 'Ta' and 't' are time lags due to
   internal computations and are kept to a minimum.

The Start message (FAH), is treated by MIDI as if it were a command comprised of a Song Position Pointer value of 0 plus a continue message (FBH).

Since the Start message and the Continue message can be received while the sequencer has been stopped by a Stop message (FCH), the sequencer should be able to start quickly in response to a Start message, even if the sequencer is in the middle of a song.

Song Position Pointer messages should be ignored if the instrument is not in MIDI sync mode (see System Real Time messages section for details on MIDI sync).

RECOMMENDED USE OF SONG POSITION POINTER

Previously it was recommended that a device wait 5 seconds after transmitting a Song Position Pointer message before it transmitted a Continue message and resumed sending MIDI Clocks. However, it is now recommended that any device receiving a Song Position Pointer (SPP) message be able to correctly receive a Continue message and subsequent MIDI Clocks while it is in the process of locating to the new position in the song. Upon locating to the new position the device must then play in sync with the device transmitting the SPP.

For example, if the transmitter sends an SPP message with a value of 4 (24 MIDI Clocks), and while locating receives a Continue as well as an additional 3 MIDI Clocks, the receiving device should begin from the 27th clock in the song.

SONG SELECT

Specifies which song or sequence is to be played upon receipt of a Start message in sequencers and drum machines capable of holding multiple songs or sequences. This message should be ignored if the receiver is not set to respond to incoming Real Time messages (MIDI Sync).

RECEPTION OF SONG POSITION AND SONG SELECT

When a device receives and recognizes a Song Position or Song Select message, it can take a relatively long time to implement the command. The receiver must continue to accept MIDI clocks after a Start has been received, and increment its Song Position while it is computing and locating to the correct address in memory for playback. For example, if a Song Position Pointer message is received which contains a value of 4 (24 MIDI Clocks), and during the process of locating a Continue and 3 clocks are received, the device should start playing from the point in its internal sequence corresponding to the 27th clock. If a Timing Clock message is missed while the receiver is dealing with Song Position, the receiver may not synchronize correctly. Song Position or Song Select messages may only be sent when the system is not playing.

TUNE REQUEST

Used with analog synthesizers to request that all oscillators be tuned.

EOX

Used as a flag to indicate the end of a System Exclusive transmission. A System Exclusive message starts with F0H and can continue for any number of bytes. The receiver will continue to wait for data until an EOX message (F7H) or any other non-Real Time status byte is received.

To avoid hanging a system, a transmitter should send a status byte immediately after the end of an Exclusive transmission so the receiver can return to normal operation. Although any Status Byte (except Real-Time) will end an exclusive message, an EOX should always be sent at the end of a System Exclusive message. Real time messages may be inserted between data bytes of an Exclusive message in order to maintain synchronization, and can not be used to terminate an exclusive message.

SYSTEM REAL TIME MESSAGES

Timing Clock	F8H
Start	FAH
Continue	FBH
Stop	FCH
Active Sensing	FEH
System Reset	FFH

System Real Time messages are used to synchronize clock-based MIDI equipment. These messages serve as uniform timing information and do not have channel numbers.

Real Time messages can be sent at any time and may be inserted anywhere in a MIDI data stream, including between Status and Data bytes of any other MIDI messages. Giving Real-Time messages high priority allows synchronization to be maintained while other operations are being carried out.

As most keyboard instruments do not have any use for Real-Time messages, such instruments should ignore them. It is especially important that Real-Time messages do not interrupt or affect the Running Status buffer. A Real-Time message should not be interpreted by a receiver as a new status.

TIMING CLOCK: Clock-based MIDI systems are synchronized with this message, which is sent at a rate of 24 per quarter note. If Timing Clocks (F8H) are sent during idle time they should be sent at the current tempo setting of the transmitter even while it is not playing. Receivers which are slaved to incoming Real Time messages (MIDI Sync mode) can thus phase lock their internal clocks while waiting for a Start (FAH) or Continue (FBH) command.

START: Start (FAH) is sent when a PLAY button on the master (sequencer or drum machine) is pressed. This message commands all receivers which are slaved to incoming Real Time messages (MIDI Sync mode) to start at the beginning of the song or sequence.

CONTINUE: Continue (FBH) is sent when a CONTINUE button is hit. A sequence will continue from its current location upon receipt of the next Timing Clock (F8H).

STOP: Stop (FCH) is sent when a STOP button is hit. Playback in a receiver should stop immediately.

START OR CONTINUE MESSAGES

When a receiver is synchronized to incoming Real Time messages (MIDI Sync mode), the receipt of a Start (FAH) or Continue (FBH) message does not start the sequence until the next Timing Clock (F8H) is received. The FA and F8 should be sent with at least 1 millisecond time between them so the receiver has time to respond. However, a receiver should be able to respond immediately to the first F8H after receiving the Start or Continue.

Neither the transmitter
nor the receiver advances
during this interval
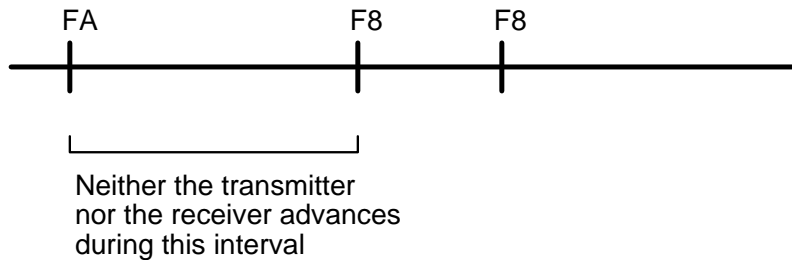
When the receiver is operating off of its internal clock it may ignore all Start, Stop and Continue messages or it may respond to these messages and start, stop or continue playing according to its own internal clock when these messages are received over MIDI. This decision is left up to the designer.

STOP MESSAGE

When a master sequencer is stopped it should send out the Stop message (FCH) immediately, so that any other devices slaved to it will also stop. The sequencer's internal location should be set as it was in when Stop was sent. This way, if Continue is pressed, all instruments connected to the master will continue from the same point in the song without need for a Song Position Pointer message.

Upon receiving a Stop message (FCH), a receiver should stop playing and not increment its Song Position when subsequent Timing Clock messages are received. The current Song Position should be maintained in the event that a Continue is received and the sequence is continued from the point that it was stopped. If a Song Position Pointer message is received, the device should change its internal Song Position and prepare to begin playback from the new location.

If any Note-Off messages have not been sent for corresponding Note-Ons sent before Stop was pressed, the transmitter should send the correct Note-Off messages to shut off those notes. An All Notes Off message can also be sent, but this should not be sent in lieu of the corresponding Note-Off messages as not every instrument responds to the All Notes Off message. In addition to note events, any controllers not in their initialized position (pitch wheels, sustain pedal, etc.) should be returned to their normal positions.

The following illustration shows a method to keep correct synchronization. These examples use an internal timebase of 96 pulses per 1/4 note, or 4 internal clocks per MIDI clock (F8H).
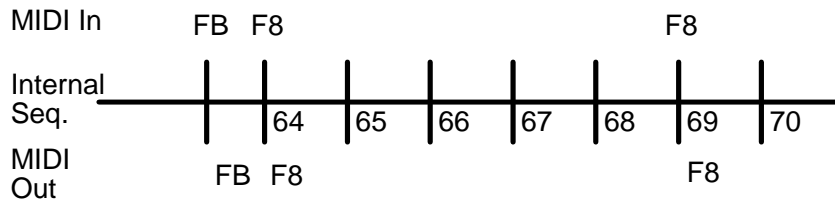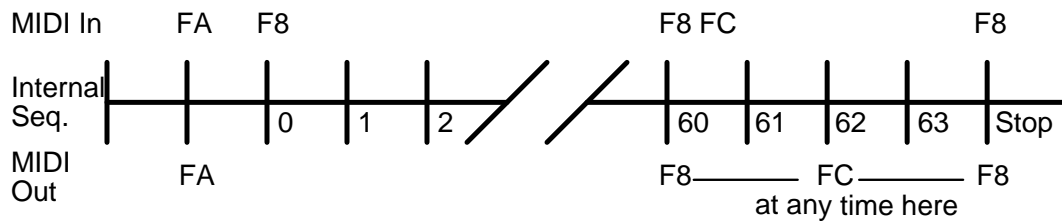
RELATIONSHIP BETWEEN CLOCKS AND COMMANDS

A sequencer may echo incoming timing and voice information out the MIDI Out port while playing its own sequenced parts. System Real Time messages should always be given time priority when the data is merged in this manner. To accomplish this, it is permissible to change the actual order of bytes to accommodate Real Time messages. However, all Real Time bytes (F8H, FAH, FBH, FCH) must be sent in the order in which they are received.

In the example below, a Note-On message is delayed slightly in order to give a priority to sending an F8H.
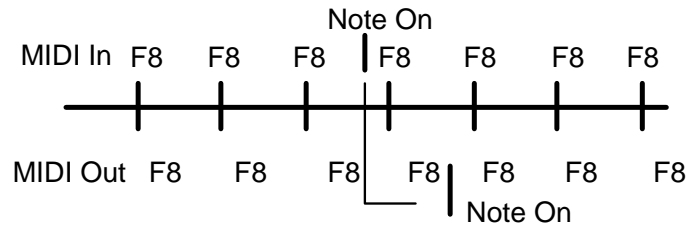
In order to avoid displacing clock messages in time, in addition to reversing their order with a voice message (as shown above), they may be also be inserted between the bytes of voice, common, or other messages. At no time should either an incoming clock byte or any voice message be dropped, but their order can be changed to accommodate the need for accurate timing.

A sequencer may continue sending timing clock (F8H) while it is stopped. The advantage of this is that a slaved device can know the starting tempo of a sequence just as the Start command is received.

PRIORITY OF COMMANDS

Redundant commands, such as receiving a Stop command while already stopped, or a Start or Continue command while already playing, should simply be ignored.

If a clock based device receives commands both from its front panel and via its MIDI In, priority should be given to the most recently received command. However, it is also acceptable for a device to ignore either its front panel or incoming Real Time commands depending on its current operating mode. For example, a device set to respond to incoming MIDI clocks and Real Time commands may ignore the commands received from its front panel. It may also ignore incoming Real Time commands while set to operate with its internal clock.

ACTIVE SENSING

Use of Active Sensing is optional for either receivers or transmitters. This byte (FE) is sent every 300 ms (maximum) whenever there is no other MIDI data being transmitted. If a device never receives Active Sensing it should operate normally. However, once the receiver recognizes Active Sensing (FE), it then will expect to get a message of some kind every 300 milliseconds. If no messages are received within this time period the receiver will assume the MIDI cable has been disconnected for some reason and should turn off all voices and return to normal operation. It is recommended that transmitters transmit Active Sensing within 270ms and receivers judge at over 330ms leaving a margin of roughly 10%.

The following flowchart shows the correct method of implementing Active Sensing:

MIDI
Reception
Interrupt

Receive Data

Clear Timer

≠FE

FE

Set Flag

Exit

To
Other
Operaton

Timer
Interrupt

Flag 0

Increment
Timer

Exit

≤ 300 ms

> 300 ms

Exit

Turn All
Notes Off

Reset Flag

Exit
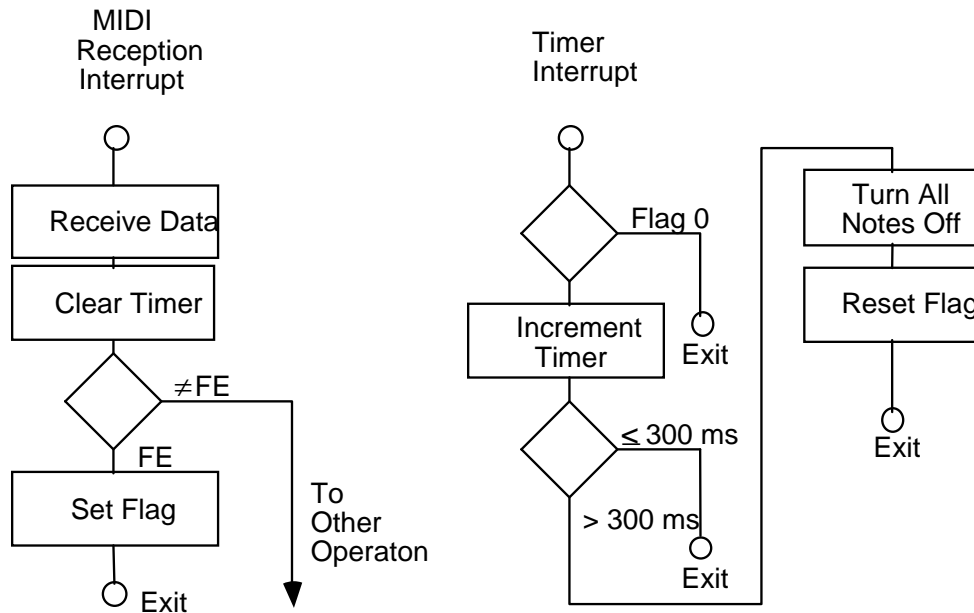
SYSTEM RESET

System Reset commands all devices in a system to return to their initialized, power-up condition. This message should be used sparingly, and should typically be sent by manual control only. It should <u>not</u> be sent automatically upon power-up and under no condition should this message be echoed.

If System Reset is recognized, the following operations should be carried out:

    1) Set Omni On, Poly mode (if implemented)
    2) Set Local On
    3) Turn Voices Off
    4) Reset all controllers
    4) Set Song Position to 0
    5) Stop playback
    6) Clear Running Status
    7) Reset the instrument to its power-up condition

SYSTEM EXCLUSIVE MESSAGES

System Exclusive	F0H

System messages are not assigned to any particular MIDI channel. Thus, they will be recognized by MIDI receivers regardless of the basic channel to which they are set. System Exclusive messages, however, have a different purpose. Each instrument's System Exclusive messages (hereafter abbreviated as "Exclusive" messages) have their own special format according to an assigned manufacturer's ID number.

Exclusive messages are used to send data such as patch parameters, sampler data, or a sequencer memory bulk dump. A format which is appropriate to the particular type of transmitter and receiver is required. For example, an Exclusive message which sets the feedback level for an operator in an FM digital synthesizer will have no corresponding or meaningful function in an analog synthesizer.

Since the purpose of MIDI is to connect many kinds of musical instruments and peripheral equipment, it is best not to use Exclusive messages to convey real-time performance information (with the exception of special Universal messages described below). Performance information is best sent via Channel Voice messages in real time. Receivers should ignore non-universal Exclusive messages with ID numbers that do not correspond to their own ID.

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

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

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

DISTRIBUTION OF ID NUMBERS

	American	European	Japanese	Other	Special
1 byte ID:	01 – 1F	20 – 3F	40 – 5F	60 – 7C	7D – 7F
3 byte ID:	00 00 01	00 20 00	00 40 00	00 60 00	
	00 1F 7F	00 3F 7F	00 5F 7F	00 7F 7F	

00 and 00 00 00 are not to be used. Special ID 7D is reserved for non-commercial use (e.g. schools, research, etc.) and is not to be used on any product released to the public. Since Non-Commercial codes would not be seen or used by an ordinary user, there is no standard format. Special IDs 7E and 7F are the Universal System Exclusive IDs..

UNIVERSAL SYSTEM EXCLUSIVE

System Exclusive ID numbers `7E` (Non-Real Time) and `7F` (Real Time) are Universal Exclusive IDs, used for extensions to the MIDI specification. The standardized format for both Real Time and Non-Real Time Universal Exclusive messages is as follows:

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

The `<device ID>` and `<sub-ID#1>` `<sub-ID#2>` fields are described <u>in context</u> below. A complete listing of the assigned Real time and Non-Real Time messages is given in TABLE VIIa.

DEVICE ID

Since System Exclusive messages are not assigned to a MIDI Channel, the Device ID (formerly referred to as the "channel" byte) is intended to indicate which device in the system is supposed to respond. The device ID `7F`, sometimes referred to as the 'all call' device ID, is used to indicate that all devices should respond.

In most cases, the Device ID should refer to the physical device being addressed (the "hunk of metal and plastic" is a common term that has been used), as opposed to having the same meaning as channel or referring to a virtual device inside a physical device. For reference, this also corresponds to old USI discussions that included a "Unit ID" that was supposed to be attached to one UART and set of in/out ports.

However, there are exceptions - for example, what Device ID to use for a dual-transport tape deck and MMC commands? Some may feel more comfortable thinking of the Device ID as an "address" and allow for the possibility that a single physical unit may be powerful enough to have more than one valid address. (This also has more relevance as devices move from stand-alone units to cards in a computer.)

Therefore, Device ID is meant to refer to a single physical device or I/O port as a default. Sophisticated devices - such as multi-transport tape decks, computers with card slots, or even networks of devices - may have more than one Device ID, and such occurrences should be explained to the user clearly in the manual. From one to sixteen virtual devices may be accessed at each Device ID by use of the normal MIDI channel numbers, depending on the capabilities of the device.

SAMPLE DUMP STANDARD

A standard has been developed for sampler data dumps. It has been designed to work as an open or closed loop system. The closed loop system implements handshaking to improve speed and error recovery. This also accommodates machines that may need more time to process incoming data. The open loop system may be desired by those wishing to implement a simplified version with no handshaking.

Five of the basic messages are generic handshaking messages (ACK, NAK, Wait, Cancel & EOF), which are also used in other applications – for example File Dump. The remaining messages are Dump Request, Dump Header, Data Packets, and a Sample Dump Extensions message. The data formats are given in hexadecimal.

GENERIC HANDSHAKING MESSAGES

ACK:

```
F0 7E <device ID> 7F pp F7
```

        pp   Packet number

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

NAK:

```
F0 7E <device ID> 7E pp F7
```

        pp   Packet number

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

CANCEL:

```
F0 7E <device ID> 7D pp F7
```

        pp   Packet number

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

WAIT:

```
F0 7E <device ID> 7C pp F7
```

        pp   Packet number

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

EOF:

```
F0 7E <device ID> 7B pp F7
```

        pp   packet number (ignored)

This is a new generic handshaking flag which was added for the File Dump extension, and is described fully under the File Dump heading.

DUMP HEADER

```
F0 7E <device ID> 01 ss ss ee ff ff ff gg gg gg hh hh hh ii ii ii jj F7
```

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

DUMP REQUEST

```
F0 7E <device ID> 03 ss ss F7
```

ss ss	Requested sample, LSB first

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

DATA PACKET

```
F0 7E <device ID> 02 kk <120 bytes> ll F7
```

kk	Running packet count (0-127)
ll	Checksum (XOR of 7E <device ID> 02 kk <120 bytes>)

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

SAMPLE DUMP EXTENSIONS

All future extensions to the Sample Dump Standard will appear under the Sub-ID#1 (05) of the Universal System Exclusive Non-Real Time message.

MULTIPLE LOOP POINT MESSAGES:

These messages were added as an extension to the Sample Dump Standard, allowing for the definition of up to 16,383 pairs of loop points per sample. This cures the shortcoming of the Sample Dump Standard allowing only 1 pair of loop points to be defined per sample. It also allows modification of loop points without also having to send the sample itself.

The formats of these messages are as follows:

Loop Point Transmission (17 bytes):

```
F0 7E <device ID> 05 01 ss ss bb bb cc dd dd dd ee ee ee F7
```

`F0 7E <device ID>`	Universal System Exclusive Non-Real Time header
`05`	Sample Dump Extensions (sub-ID#1)
`01`	Multiple Loop messages (sub-ID#2)
`ss ss`	Sample Number (LSB first)
`bb bb`	Loop number (LSB first; 7F 7F = delete all loops)
`cc`	Loop type
	00 = Forwards Only (unidirectional)
	01 = Backwards/Forwards (bi-directional)
	7F = Off
`dd dd dd`	Loop start address (in samples; LSB first)
`ee ee ee`	Loop end address (in samples; LSB first)
`F7`	EOX

Loop Points Request (10 bytes):

```
F0 7E <device ID> 05 02 ss ss bb bb F7
```

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

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

SAMPLE DUMP TRANSMISSION SCENARIO

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

A data packet consists of its own header, a packet number, 120 data bytes, a checksum, and an End Of Exclusive (EOX). The packet number starts at 00 and increments with each new packet, resetting to 00 after it reaches 7FH. This is used by the receiver to distinguish between a new data packet and one being resent. This number is followed by 120 bytes of data which form 30, 40 or 60 words (MSB first) depending on the sample format.

Each data byte consists of 7 bits. If the sample format is 8-14 bit, two bytes form a word. Sample formats of 15-21 bits require three bytes/word (yielding 40 words/packet). Sample formats of 22-28 bits require four bytes/word (yielding 30 words/packet). Information is left-justified within the 7-bit bytes and unused bits are filled in with zeros. For example, the sample word FFFH would be sent as 01111111B 01111100B. The word FFFH represent a full positive value (000H represents full negative). The checksum is the XOR of 7E `<device ID>` 02 <packet number> <120 bytes>.

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

When a sampler is sending a data dump, it should send a packet and watch its MIDI In port. If an ACK is received, it sends the next packet. If a NAK is received and the packet number matches that of the previous packet, it re-sends that packet. If the packet numbers do not match and the sampler has no facility to send packets out of sequence, it should ignore the NAK. If a Wait is received, the sampler should watch its MIDI IN port indefinitely for another message and process it like a normal ACK, NAK, Cancel, or illegal message (which would usually abort the dump). If nothing is received within 20ms, the sampler can assume an open loop and send the next packet.

The packet numbers are included in the handshaking flags (ACK, NAK, Cancel, Wait) in order to accommodate future machines that might have the intelligence to re-transmit specific packets out of sequence (i.e. after subsequent packets have been received).

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

DEVICE INQUIRY

The following two messages are used for device identification, and are categorized as Non-Real Time System Exclusive General Information messages (sub-ID#1 = 06).

The format of the inquiry message is as follows:

```
F0 7E <device ID> 06 01 F7
```

F0 7E <device ID>	Universal System Exclusive Non-real time header
06	General Information (sub-ID#1)
01	Identity Request (sub-ID#2)
F7	EOX

A device which receives the above message would respond as follows:

(Note that if <device ID> = 7FH then the device should respond regardless of what <device ID> it is set to.)

```
F0 7E <device ID> 06 02 mm ff ff dd dd ss ss ss ss F7
```

F0 7E <device ID>	Universal System Exclusive Non-real time header
06	General Information (sub-ID#1)
02	Identity Reply (sub-ID#2)
mm	Manufacturers System Exclusive id code
ff ff	Device family code (14 bits, LSB first)
dd dd	Device family member code (14 bits, LSB first)
ss ss ss ss	Software revision level. Format device specific
F7	EOX

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

FILE DUMP

File Dump provides a protocol for transmitting files from one computer to another using MIDI. There are two primary motivations for this protocol: transmitting MIDI Files (especially tempo maps) between computers and small ROM/microcomputer-based "boxes"; and transmitting files of any type, including MIDI files, between two computers of different types. The filename is sent with the file, so that several files may be sent one after another with as little user interaction as necessary.

All File Dump messages are Exclusive Non-Real Time messages (sub-ID#1 = `07`), and begin with the following header:

```
F0 7E <device ID> 07 <sub-ID#2> ss ...
```

> `<device ID>`   device ID of message destination (`7F` is also acceptable here)
> `07`   File Dump (sub-ID#1)
> `<sub-ID#2>`   file dump message type:
>   `01`   Header
>   `02`   Data Packet
>   `03`   Request
> `ss`   device ID of message source (`7F` "all-call" is NOT acceptable here)

The source device ID is included so that it may be used by the receiver of this message in all packets which it sends back to the sender of this message. In other words, if the handshake of this transfer is between device A and device B, all messages going from A to B specify B as the destination of the message, and all messages going back from B to A specify A as the destination of the message. In order to do this, the first message to B must also specify A as the source of the first message, so that B knows the device ID of who to respond to for all response messages.

REQUEST

```
F0 7E <device ID> 07 03 ss <type> <NAME> F7
```

> `<device ID>`   device ID of request destination (will become file sender)
> `ss`   device ID of requester (will become file receiver)
> `<type>`   four 7-bit ASCII bytes: type of file
> `<NAME>`   filename: 7-bit ASCII bytes terminated by the message's F7

`<type>` describes what type of file, in a general sense, is being requested. Only the types shown below should be used; you should only use any other type if you know that the receiver will recognize it. If the device receiving a request doesn't support the requested type, it should send the Cancel message described below.

`<type>`	Recommended DOS Extension	Description
MIDI	MID	It's a MIDI File
MIEX	MEX	It's a MIDIEX File
ESEQ	ESQ	It's an ESEQ File
TEXT	TXT	It's a 7-bit ASCII Text File
BIN<space>	BIN	It's a binary file (such as any MS-DOS file)
MAC<space>	MAC	It's a Macintosh file (with MacBinary header)

If `<type>` is MAC, this means a Macintosh file is being requested. Because Macintosh files contain two "forks," and other important Finder information, they are sent as their MacBinary image. Note that programs wishing to transmit only MIDI Files, even on the Macintosh, won't need to worry about

MacBinary, because MIDI Files must always use the MIDI designation to be universally recognized as MIDI Files.

The filename may be any length, and may be omitted entirely. If it is omitted, it means "whatever is currently loaded." The filename may contain only printable ASCII characters (20H through 7EH). Colons and backslashes may optionally be interpreted as path specifiers: these characters should be avoided in filenames if this behavior is not desired by the user. If the device receiving the request message does not have a file system, it should send whatever is currently loaded, using a null filename.

If the device receiving the request message is a computer, it should initiate a transfer if it recognizes the filename, or if there is no filename but there is a "currently loaded" file. If these conditions aren't met, it may either prompt the user for a valid filename (displaying the filename supplied in the dump message), or just send the Cancel message back to the requester. If the user is to be prompted, the Wait message should be sent to the requester so that it knows that it may be awhile before the transfer is initiated or a Cancel is to be sent.

HEADER

```
F0 7E <device ID> 07 01 ss <type> <length> <NAME> F7

        <device ID>  device ID of requester/receiver
        ss           device ID of sender
        <type>       four 7-bit ASCII bytes: type of file
        <length>     four 7-bit bytes: actual (un-encoded) file length, LSB first
        <NAME>       filename: 7-bit ASCII bytes terminated by the message's F7
```

`<type>` and `<NAME>` are exactly as described in the Dump Request message.

If the length of the file is not known (because it will be converted on the fly), zero may be sent as the length.

If the sender is a small ROM-based "box" without files, it need not send a filename. If it is a computer, and there is a filename associated with it, it should be sent in the header. As described above, it may be any length, must only contain printable ASCII characters, and may contain path description characters. For maximum compatibility, no path information should be sent. DOS-like machines should send the file extension as part of the name, separated by a period, with no trailing spaces before the period.

If the receiver is a computer, and if the program running supports receiving files, it should modify the filename if necessary to make it appropriate for its file system. For instance, if it is a DOS machine, and the given filename contains a period, it should interpret everything after the period as the file's extension. If there is no period, it should use the appropriate extension listed above. If it is running interactively, it should prompt the user with the filename supplied in the dump message, so that the user can modify it if desired, if no name is sent, or if a file by that name already exists. If the user is to be prompted, the Wait message should be sent to the sender so that it knows that it may be awhile before the transfer is continued or a Cancel is sent.

If the receiver is a small ROM-based "box" without files, or a program on a computer which only expects this protocol to replace the file currently in memory, it should simply ignore the filename and replace its current memory contents with the contents of the transmitted file, if the file is a supported type.

DATA PACKET

```
F0 7E <device ID> 07 02 <packet #> <byte count> <data> <chksm> F7

        <device ID>           device ID of receiver
```

<pre>   <packet #>   one-byte packet count
    <byte count>   one-byte packet size: number of encoded data bytes
                        minus one
    <data>   the data, encoded as described below
    <chksm>   one-byte checksum (XOR of all bytes which
                        follow F0 up to the checksum byte, similar to
                        sample dump)</pre>

The total size of a data packet may be slightly larger than for sample dump: 137 bytes maximum. The packet number starts at 00 and increments with each new packet, resetting to 00 after it reaches 7FH. This is used by the receiver to detect missed packets. The byte count is the number of encoded data bytes minus one: for example, 64 stored bytes of the file are encoded in 74 transmitted bytes (as described below): the byte count would be 73. (Subtracting one allows sending 128 transmitted data bytes: one would never need to send zero bytes).

Instead of nibblizing, which would double transmission time, the data is "7-bit-ized" so that the transmission time is more like 12% more than sending it as 8-bit (which isn't possible over MIDI). Each group of seven stored bytes is transmitted as eight bytes. First, the sign bits of the seven bytes are sent, followed by the low-order 7 bits of each byte. (The reasoning is that this would make the auxiliary bytes appear in every 8th byte without exception, which would therefore be slightly easier for the receiver to decode.) The seven bytes:

<pre>   AAAAaaaa BBBBbbbb CCCCcccc DDDDdddd EEEEeeee FFFFffff GGGGgggg</pre>

are sent as:

<pre>   0ABCDEFG
    0AAAaaaa 0BBBbbbb 0CCCcccc 0DDDdddd 0EEEeeee 0FFFffff 0GGGgggg</pre>

From a buffer to be encoded, complete groups of seven bytes are encoded into groups of eight bytes. If the buffer size is not a multiple of seven, there will be some number of bytes left over after the groups of seven are encoded. This short group is transmitted similarly, with the sign bits occupying the most significant bits of the first transmitted byte. For example:

<pre>   AAAAaaaa BBBBbbbb CCCCcccc</pre>

are transmitted as:

<pre>   0ABC0000 0AAAaaaa 0BBBbbbb 0CCCcccc</pre>

Since the maximum packet size is 128 transmitted bytes, this corresponds to sixteen groups of seven bytes, or 112 stored bytes.

HANDSHAKING FLAGS

For handshaking messages, the same generic set originally created for Sample Dump Standard – plus a new EOF message – are to be used (Non-Real Time sub-ID#1 = 7B-7F). Since these first four message were explained in the Sample Dump section, only newly significant information will be presented here.

```
F0 7E <device ID> <sub-ID#1> pp F7
```

        `<device ID>` device ID of packet sender (message destination)
        `<sub-ID#1>` handshake message:

	7B	End of File
	7C	Wait
	7D	Cancel
	7E	NAK
	7F	ACK

    `pp`   packet number

NAK:

```
F0 7E <device ID> 7E pp F7
```

        `<device ID>` device ID of packet sender (ACK receiver)
        `pp`   packet number

This should be sent whenever the length of a message was wrong, or the checksum was incorrect. After receiving a NAK, the sender should resend the packet. After sending a NAK, the receiver should expect the same packet to be resent. If the same packet has an error three consecutive times, a Cancel should be sent instead of a NAK. If the packet number was wrong, such as if a packet (or a NAK) was missed, the Cancel message should be sent instead of a NAK.

ACK:

```
F0 7E <device ID> 7F pp F7
```

        `<device ID>` device ID of packet sender (ACK receiver)
        `pp`   packet number

The packet number represents the packet being acknowledged as correct. The packet number in the ACK responding to the Header is undefined.

WAIT:

```
F0 7E <device ID> 7C pp F7
```

        `<device ID>` device ID of Wait receiver
        `pp`   packet number (ignored)

This handshaking flag is used after receiving a File Header, Data Packet, or File Dump Request. When responding to a Header it means "Do not send data packets until you receive an ACK (or a Cancel)." When responding to a data packet, it means "Do not send any more packets until you receive an ACK or NAK (or a Cancel)." When used in response to a File Dump Request, it means "Your File Header (or a Cancel) will follow soon – be patient."

This message is important for systems in which the receiver may need to perform other operations, such as disk access or prompting the user, before processing the remainder of the dump. A slow device may in fact wish to transmit a Wait every time it receives a File Header, Data Packet, or File Dump Request, thus giving itself unlimited time in which to digest the received data and respond appropriately.

CANCEL:

```
F0 7E <device ID> 7D pp F7
```

> <device ID> device ID of Cancel receiver
> pp   packet number (ignored)

This handshaking flag may be used at any time. It means "Abort dump." The packet number represents the packet on which the abort takes place, but is ignored by the receiver. This may be sent by either the sender or receiver when any error is detected, such as incorrect packet numbers in a data packet or a handshake message; or when a dump is canceled by the user. If the sender aborts a transmission, it should use the receiver's device ID in the Cancel message (which it put in the header (<device ID>) in the first place). If the receiver aborts a transmission, it should use the sender's device ID in the Cancel message (which the sender put in the header (ss)).ACK

END OF FILE (EOF):

```
F0 7E <device ID> 7B pp F7
```

> <device ID> device ID of receiver
> pp   packet number (ignored)

This is the fifth generic handshaking flag within MIDI, sub-ID#1 (7B). After sending the last packet of a lengthy message (such as File Dump), the sender must send an EOF message to inform the receiver that the entire file has been sent. This is critical if the length in the File Dump Header is 0 (which means that the file length is unknown), because this is the only way the receiver can know the transmission is complete and correct. This message must be sent even if the correct length is known at the beginning. EOF requires no response from the receiver.

FILE DUMP TRANSMISSION SCENARIO

The File Dump Request is optional. A device may request a file (or memory contents), using the Request message, or a user may initiate a file dump without a request message being sent. Within 200 msec after receiving the Request message EOX (F7), the sender must respond with a File Dump header, Wait, or Cancel. If it responds with Wait, it may send a File Dump header or a Cancel message whenever it's good and ready.

The sender then sends a File Dump header message. Within 200 msec after receiving the Header EOX (F7), the receiver must respond with ACK, Wait, or Cancel. If it responds with Wait, it may send an ACK or a Cancel message whenever it's good and ready. If the sender does not receive any message during this time, it assumes open loop transmission, and proceeds as if an ACK had been received.

The sender then sends a data packet. As the receiver receives the data packet, it keeps a running checksum. If the checksums match, and it can deal with the data immediately, it sends an ACK and waits for the next packet. If it needs more than 50 msec to store the data, it sends a Wait message. (After storing the data, it then sends an ACK to continue the process). If the checksums do not match, or if the length is wrong, it sends a NAK and waits for the same packet to be resent. If the packet number is not the one it was expecting, it sends a Cancel message and ignores all further data packets until a

new header is sent (in the open-loop case, the sender won't ever receive a Cancel message). If the receiver's memory ever becomes full, even during the last packet, it should send a Cancel to the sender.

When a device is sending a data dump, it should send a packet and watch its MIDI IN port. If an ACK is received, it should send the next packet immediately. If a NAK is received and the packet number matches that of the previous packet, it re-sends that packet. If the packet number of an ACK or a NAK do not match the number of the packet just sent, the sender should send a Cancel message, and abort the transmission. If a Wait is received, it should watch its MIDI IN port indefinitely for another message. If it receives an ACK or NAK, it should process it normally, and continue; if it receives a Cancel or an illegal message, it should abort the dump process. If nothing is received in 50 msec after a data packet or 200 msec after a header, it can assume an open loop and send the next packet.

After the receiver ACKs the last packet, the sender transmits an EOF. No ACK is required for this message. The file dump is then complete.

Any packet may contain any number of bytes, up to 128 encoded data bytes. Most devices probably will transmit several packets of equal size, and send what's left over as a final packet. However, the receiver should never make any assumption about packet size.

MIDI TUNING

This is an addition to the MIDI specification which allows the sharing of "microtunings" (user-defined scales other than 12-tone equal temperament) among instruments of different manufacture, and the switching of these tunings during real-time performance.

The messages include:

> – bulk tuning dump request (non-real-time)
> – bulk tuning dump (non-real-time)
> – single-note tuning change (<u>real-time</u>)

Even though the first two messages are in the Universal Non-Real Time area and the last in the Real Time area, they keep the same sub-IDs to more obviously group them and possibly ease the parsing of them. Single Note Retuning is a part of the proposal which allows retuning of individual MIDI note numbers to new frequencies in real time as a performance control.

The standard does not attempt to dictate how a manufacturer implements microtuning, but provides a general means of sharing tuning data among different instruments.

This goal does require shared assumptions which have some architectural implications. The standard requires that any of the 128 defined MIDI key numbers (or at least those MIDI key numbers covered by the instrument's playable range) be tunable to any frequency within the proposed frequency range. The standard also strongly suggests, but does not enforce, an exponential (constant cents) rather than linear (constant Hertz) tuning resolution across the instrument's frequency range.

The standard permits the changing of tunings in real-time, both by the selection of presets and on a per-note basis. When a sounding note is affected by either real-time tuning message, the note should instantly be re-tuned to the new frequency while it continues to sound; this change should occur without glitching, forced Note-Offs, re-triggering or other audible artifacts (see section 4, "Additional").

The standard provides for 128 tuning memory locations (programs). As with the MIDI program change message, this is a maximum value. An instrument supporting the standard may have any lesser number of tuning programs. The standard requires only that all existing tuning programs respond to the messages as specified (See section 3, "Continuous Controller Messages").

Although directly applicable to some existing instruments, the standard attempts to define a coherent framework within which the designers of future instruments can profitably work. It is hoped that by providing this framework the standard will make microtunability more easily implemented and more common on MIDI instruments.

FREQUENCY DATA FORMAT

The frequency resolution of the standard should be stringent enough to satisfy most demands of music and experimentation. The standard provides resolution somewhat finer than one-hundredth of a cent. Instruments may support the standard without necessarily providing this resolution in their hardware; the standard simply permits the transfer of tuning data at any resolution up to this limit.

Frequency data shall be sent via system exclusive messages. Because system exclusive data bytes have their high bit set low, containing 7 bits of data, a 3-byte (21-bit) frequency data word is used for specifying a frequency with the suggested resolution. An instrument which does not support the full suggested resolution may discard any unneeded lower bits on reception, but it is preferred where possible that full resolution be stored internally, for possible transmission to other instruments which can use the increased resolution.

Frequency data shall be defined in units which are fractions of a semitone. The frequency range starts at MIDI note 0, C = 8.1758 Hz, and extends above MIDI note 127, G = 12543.875 Hz. The first byte of the frequency data word specifies the nearest equal-tempered semitone below the frequency. The next two bytes (14 bits) specify the fraction of 100 cents above the semitone at which the frequency lies. Effective resolution = 100 cents / 2^{14} = .0061 cents.

One of these values (7F 7F 7F) is reserved to indicate not frequency data but a "no change" condition. When an instrument receives these bytes as frequency data, it should make no change to its stored frequency data for that MIDI key number. This is to prevent instruments which do not use the full range of 128 MIDI key numbers from sending erroneous tuning data to instrument which do use the full range. The three-byte frequency representation may be interpreted as follows:

```
0xxxxxxx   0abcdefg   0hijklmn
```

```
xxxxxxx              = semitone
abcdefghijklmn       = fraction of semitone, in .0061-cent units
```

Examples of frequency data:

```
00 00 00 =        8.1758 Hz    (C – normal tuning of MIDI key no. 0)
00 00 01 =        8.2104 Hz
01 00 00 =        8.6620 Hz
0C 00 00 =       16.3516 Hz
3C 00 00 =      261.6256 Hz    (middle C)
3D 00 00 =      277.1827 Hz    (C# – normal tuning of MIDI key no. 61)
42 7F 7F =      439.9984 Hz
43 00 00 =      440.0000 Hz    (A-440)
43 00 01 =      440.0016 Hz
78 00 00 =     8372.0190 Hz    (C – normal tuning of MIDI key no. 120)
78 00 01 =     8372.0630 Hz
7F 00 00 =    12543.8800 Hz    (G – normal tuning of MIDI key no. 127)
7F 00 01 =    12543.9200 Hz
7F 7F 7E =    13289.7300 Hz    (top of range)
7F 7F 7F =        no change    (reserved)
```

BULK TUNING DUMP REQUEST

A bulk tuning dump request is as follows:

```
F0 7E <device ID> 08 00 tt F7
```

```
F0 7E         Universal Non-Real Time SysEx header
<device ID>   ID of target device
08            sub-ID#1 = MIDI Tuning Standard
00            sub-ID#2 = 00H, bulk dump request)
tt            tuning program number (0 – 127)
F7            EOX
```

The receiving instrument shall respond by sending the bulk tuning dump message described in the following section for the tuning number addressed.

BULK TUNING DUMP

A bulk tuning dump comprises frequency data in the 3-byte format outlined in section 1, for all 128 MIDI key numbers, in order from note 0 (earliest sent) to note 127 (latest sent), enclosed by a system exclusive header and tail. This message is sent by the receiving instrument in response to a tuning dump request.

```
F0 7E <device ID> 08 01 tt <tuning name> [xx yy zz] ... chksum F7
```

`F0 7E`	Universal Non-Real Time SysEx header
`<device ID>`	ID of responding device
`08`	sub-ID#1 = MIDI Tuning Standard
`01`	sub-ID#2 = 01H, bulk dump reply)
`tt`	tuning program number (0 – 127)
`<tuning name>`	16 ASCII characters
`[xx yy zz]`	frequency data for one note (repeated 128 times)
`chksum`	checksum (XOR of `7E <device ID> 01 tt` <388 bytes>)
`F7`	EOX

If an instrument does not use the full range of 128 MIDI key numbers, it may ignore data associated with un-playable notes on reception, but it is preferred where possible that the full 128-key tuning be stored internally, for possible transmission to other instruments which can use the increased resolution. On transmission, it may if necessary pad frequency data associated with un-playable notes with the "no change" frequency data word defined above. For keys in the instrument's key range, the pitch that is sent should be the pitch that key would play if it were received as part of a note-on message. For keys outside the key range, `7F 7F 7F` may be sent.

SINGLE NOTE TUNING CHANGE (<u>REAL-TIME</u>)

The single note tuning change message (Exclusive Real Time sub-ID#1 = `08`) permits on-the-fly adjustments to any tuning stored in the instrument's memory. These changes should take effect immediately, and should occur without audible artifacts if any affected notes are sounding when the message is received.

```
F0 7F <device ID> 08 02 tt ll [kk xx yy zz] F7
```

`F0 7F`	Universal Real Time SysEx header
`<device ID>`	ID of target device
`08`	sub-ID#1 (MIDI Tuning Standard)
`02`	sub-ID#2 (02H, note change)
`tt`	tuning program number (0 – 127)
`ll`	number of changes (1 change = 1 set of [kk xx yy zz])
`[kk`	MIDI key number
`xx yy zz]`	frequency data for that key (repeated 'll' number of times)
`F7`	EOX

This message also permits (but does not require) multiple changes to be embedded in one message, for the purpose of maximizing bandwidth. The number of changes following is indicated by the byte `ll`; the total length of the message equals 8 + (`ll` x 4) bytes.

If an instrument does not support the full range of 128 MIDI key numbers, it should ignore data associated with un-playable notes on reception.

This message can be used to make changes in inactive (background) tunings as well. This message may also, at the discretion of the manufacturer, be transmitted by the instrument under particular circumstances (for example, while holding down one or more keys and pressing a "send-single-note-tuning" front panel button).

CHANGING TUNING PROGRAMS

A registered parameter number shall be allotted to select any of the instrument's stored tunings as the "current" or active tuning. Instruments which permit the storage of multiple microtunings should respond to this message by instantly changing the "current" tuning to the specified stored tuning. This change takes effect immediately and must occur without audible artifacts (notes-off, resets, re-triggers, glitches, etc.) if any affected notes are sounding when the message is received.

As with the MIDI program change message, no assumptions are made as to the underlying architecture of the instrument. For instance, in cases where layered or multi-timbral sounds might be assigned to different tunings, so that more than one tuning might be active, the manufacturer may decide how best to interpret this message. The basic channel number might prove useful in discriminating between multiple active tunings, or a certain range of tuning programs might be set aside and defined as active.

The message is sent as a registered parameter number controller message, followed by either a data entry, data increment, or data decrement controller message, e.g. (with running status shown):

```
Bn 64 03 65 00 06 tt      (data entry)
Bn 64 03 65 00 60 7F      (data increment)
Bn 64 03 65 00 61 7F      (data decrement)
     n      = basic channel number
     tt     = Tuning Program number (1-128)
```

Likewise, a Tuning Bank Change Registered Parameter number is also assigned as follows:

```
Bn 64 04 65 00 06 tt      (data entry)
Bn 64 04 65 00 60 7F      (data increment)
Bn 64 04 65 00 61 7F      (data decrement)
     n      = basic channel number
     tt     = Tuning Bank number (1-128)
```

For maximum flexibility, this Bank Number is kept separate from the normal Program Change Bank Select (controller #00). However, an instrument may wish to link the two as a feature for the user, especially if a tuning bank is stored alongside a patch parameter bank (for example, on a RAM cartridge).

If an instrument receives a Tuning Program or Bank number for which it has no Program or Bank, it should ignore that message. Standard mappings of "common" tunings to program numbers are not being proposed at this time.

Additional

There is some question as to whether instantaneous response to real-time tuning changes is desirable in every circumstance. In some performance situations it makes more sense if a tuning change affect only those notes which occur subsequent to the change, and not affect sounding notes. But there are also situations in which tuning changes should take place instantaneously, as specified in the standard, and should affect sounding notes without disrupting their continuity.

If the instrument responds well in the latter situation, some work-around is possible for the former. The reverse is not true. Therefore the standard requires that tuning changes immediately affect sounding notes. Manufacturers might, however, consider implementing a switchable "instantaneous/next-note-on" option within an instrument.

Single Note Retuning is intended for performance. Because of there are two primary concerns: 1) the RAM required for temporary copies of tuning tables; and 2) the computational load of smoothly updating the pitch of affected active notes. It is clear that in order to recognize the Single Note Retune message, a copy of the current Tuning Program needs to be kept in RAM. In a multi-timbral environment there is potentially a copy for each virtual instrument. A high-end instrument could afford the upwards of 8K of RAM needed for per-virtual-instrument copies. More modest instruments may choose to only implement one alterable RAM table and either make it available only to the basic channel virtual instrument or require that all instruments share the same tuning. Provided that it is explained in the user's manual, any of these methods is acceptable.

Additional information on alternate tunings:
The Just Intonation Network
MIDI Tuning Standards Committee
535 Stevenson St.
San Francisco, CA 94103

GENERAL MIDI SYSTEM MESSAGES

There is a defined set of Universal Non-Real Time SysEx messages for General MIDI (sub-ID#1 = `09`). The current messages (below) turn GM mode on/off on 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` 'All Call')
`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` 'All Call')
`09`	sub-ID#1 = General MIDI message
`02`	sub-ID#2 = General MIDI Off
`F7`	EOX

MTC FULL MESSAGE, USER BITS, REAL TIME CUEING

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

Universal System Exclusive Real Time sub-ID#1 (01) is used for the MTC Full Message, and for defining MTC User Bits. Real Time sub-ID#1 (05) is used for MIDI Cueing.

See the separate MTC Detailed Specification for complete details.

MIDI SHOW CONTROL

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.

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

See the separate MSC Detailed Specification for complete details.

NOTATION INFORMATION

Universal System Exclusive Real Time subID#1 (03) is used for communicating musical structure information in real time.

The messages include Bar Marker, Time Signature (Delayed), and Time Signature (Immediate).

BAR MARKER

The Bar Marker message specifies that the next MIDI clock received is the first clock of a measure, and thus a new bar.

The message format is as follows:

```
F0 7F <device ID> 03 01 aa aa F7
```

F0 7F	Universal Real Time SysEx Header	
<device ID>	ID of target device (default = 7F [all])	
03	sub ID#1 = Notation Information	
01	sub ID#2 = Bar Number Message	
aa aa	bar number; lsb first	
	[00 40]	not running
	[01 40] – [00 00]	count-in
	[01 00] – [7E 3F]	bar number in song
	[7F 3F]	running; bar number unknown
F7	EOX	

The numbering system uses the largest possible negative number as the "not running" flag; count-in bars are negative numbers until they reach zero, which is the last bar of count-in (systems that have only 1 bar of count-in don't have to deal with negative numbers – just count from "zero" on up); bar numbers then increment through positive numbers, with the highest positive number reserved as "running, but I don't know the bar number" (or the bar number has exceeded 8K).

If MIDI clocks (F8s) are also being sent, this bar number takes effect at the next received F8. If MTC but no MIDI clocks are being sent, this bar number takes effect at the next received F1 xx. It may be displayed as soon as received (in the event that it was sent while a drum machine or sequencer is paused, but has located to a new section of the song).

Please note that this message is intended for information and high-level synchronization as opposed to low-level synchronization, and should not be taken as a substitute for other MIDI timing messages.

The Bar Marker message is critical for other Notation messages (such as Time Signature) which have the option of taking effect immediately or on the next received Bar Marker message. In the later case, extra information can be sent at any time during the previous bar without taking effect. This will minimize clogging by allowing enough room between the last F8/F1 xx of a bar and the first F8/F1 xx of the next. With the Bar Marker being sent every bar, a receiver does not have to keep track of MIDI clocks to know exactly where it is.

Therefore, it is strongly suggested that the Bar Number be sent immediately after the last F8 or F1 xx message of the previous bar, to prevent possible clogging, jitter, and/or message transposition (MIDI mergers may also want to be sensitive to this message to prevent it getting delayed past a following F8).

TIME SIGNATURE

The Signature Messages are used to communicate a new time signature to a receiving device. There are two forms, Immediate and Delayed. The Immediate form (sub id #2 = 02H [bit 6 = reset]) takes effect upon receipt (or on the next received MIDI clock if slaved to MIDI sync). The Delayed form (sub-ID#2 = 42H [bit 6 = set]) takes effect upon the receipt of the next Bar Marker message. However, it may be displayed immediately.

Time Signature (Immediate):

```
F0 7F <device id> 03 02 ln nn dd bb cc bb [nn dd...]  F7
```

F0 7F	Universal Real Time SysEx header
<device id>	ID of target device (default = 7F [all])
03	sub-ID#1 = Notation Information
02	sub-ID#2 = Time Signature - Immediate
ln	number of data bytes to follow
nn	number of beats (numerator) of time signature
dd	beat value (denominator) of time signature (negative power of 2)
cc	number of MIDI clocks in a metronome click
bb	number of notated 32nd notes in a MIDI quarter note
[nn dd...]	additional pairs of time signatures to define a compound time signature within the same bar.
F7	EOX

Time Signature (Delayed):

```
F0 7F <device id> 03 42 ln nn dd bb cc bb [nn dd...]  F7
```

F0 7F	Universal Real Time SysEx header
<device id>	ID of target device (default = 7F [all])
03	sub-ID#1 = Notation Information
42	sub-ID#2 = Time Signature - Delayed
ln	number of data bytes to follow
nn	number of beats (numerator) of time signature
dd	beat value (denominator) of time signature (negative power of 2)
cc	number of MIDI clocks in a metronome click
bb	number of notated 32nd notes in a MIDI quarter note
[nn dd...]	additional pairs of time signatures to define a compound time signature within the same bar.
F7	EOX

The additional data in [nn dd...] must always be in pairs. If there are not additional time signatures specified, ln (the length of the data) = 4. It is incremented by multiples of 2 for every extra time signature pair that exists within the bar.

The data format here duplicates that of the Standard MIDI File Time Signature Meta Event (FF 58), with extra bytes for compound time signatures. The bytes for the compound time signatures were added at the end so that the current Meta Event could be extended to match the format of this message, while keeping the leading bytes of the event the same.

The burden is placed on the transmitter to indicate ahead of time what the time signature will be in the next bar. It is not the responsibility of the receiver to count clocks and decode it. It is intended that interpretation of the Notation family of messages be made as simple as possible for the receiver so that devices with displays (which may not be following MIDI clocks) could easily pass useful information to the user.

DEVICE CONTROL

MASTER VOLUME AND MASTER BALANCE

These messages are intended to produce the same effect as volume and balance controls on a stereo amplifier. They are intended mainly for General MIDI instruments (so that one Master Volume control can simultaneously fade out all the layers in a sound module, for example), although there may be wider applications.

Because these messages are intended to address "devices" as opposed to MIDI "channels" they have been defined as Universal Real Time System Exclusive messages (sub-ID#1 = 04). The corresponding "channel" messages are the controllers Channel Volume (formerly Main Volume) (CC number 07) and Balance (CC number 08).

Master Volume:

```
F0 7F <device id> 04 01 vv vv F7
```

F0 7F <device id>	Universal Real Time SysEx header
04	sub-ID#1 = Device Control
01	sub-ID#2 = Master Volume
vv vv	Volume (lsb first); 00 00 = volume off
F7	EOX

Master Balance:

```
F0 7F <device id> 04 02 bb bb F7
```

F0 7F <device id>	Universal Real Time SysEx header
04	sub-ID#1 = Device Control
02	sub-ID#2 = Master Balance
bb bb	Balance (lsb first); 00 00 = hard left; 7F 7F = hard right
F7	EOX

In order to properly respond to these messages and their channel-aimed counterparts, a device must internally track three volume and two balance scalars as follows:

1. Received on its own ID (which matches its knob on the front panel; if no knob or if knob is not scanned then power up default is set at full volume)

2. Received on the 'All Call' or 'broadcast' ID (7F)

3. Channel messages.

This way, each virtual/channel-based instrument can be individually mixed, then a device could be individually scaled, and then all devices could be brought down together without forgetting their individual levels.

MIDI MACHINE CONTROL

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.

MIDI Machine Control uses two Universal Real Time System Exclusive messages (sub-ID#1's), one for Commands (transmissions from Controller to Controlled Device), and one for Responses (transmissions from Controlled Device to Controller). (sub-ID#1 = `06, 07`)

See the separate MMC Detailed Specification for complete details.

ADDITIONAL EXPLANATIONS AND APPLICATION NOTES

RUNNING STATUS

Running status is a convenient short cut in transmission of data which saves time and makes it easier to minimize delays of transmitted MIDI data from the actual performance. With Running Status, after the first message, additional messages of the same type (i.e. Note On messages on the same MIDI channel) are sent without repeating the status byte for every message. Receivers must understand that if a data byte is received as the first byte of a message, the most recent, or "running" status is assumed.

For example, a note is normally played by transmitting a Note On Status Byte (90H) followed by the key number value (0kkkkkkk) and the velocity value bytes (0vvvvvvv). With Running Status, all additional notes on the same MIDI channel can be played by simply transmitting the key number and velocity bytes. As long as all following data consists of Note Ons on the same MIDI channel the Note On status byte need not be sent again.

Running Status is most useful for Note On and continuous controller messages. As notes can be turned off by sending a Note On with a velocity value of 0, long strings of note messages can be sent without sending a Status byte for each message. If the Note Off (8nH) message is used to turn notes off, a status byte must be sent.

The following is an example of Running Status. On the top is a complete data stream with one Status Byte for each pair of Data Bytes. Below that is a compressed data stream with only one Status Byte:

```
With
Status   | 90H | 3CH | 27H |   | 90H | 40H | 2BH |   | 90H | 43H | 25H |

With
Running  | 90H | 3CH | 27H |   | 40H | 2BH |   | 43H | 25H |
Status            C Note On         E Note On       G Note On
                  (Vel. = 39)       (Vel. = 43)     (Vel. = 37)

With
Status   | 90H | 3CH | 27H |   | 80H | 3CH | 40H |   | 90H | 3EH | 29H |

With
Running  | 90H | 3CH | 27H |   | 3CH | 00H |   | 3EH | 29H |
Status            C Note On         C Note Off      D Note On
                  (Vel. = 39)       (Vel. = 0)      (Vel. = 41)
```
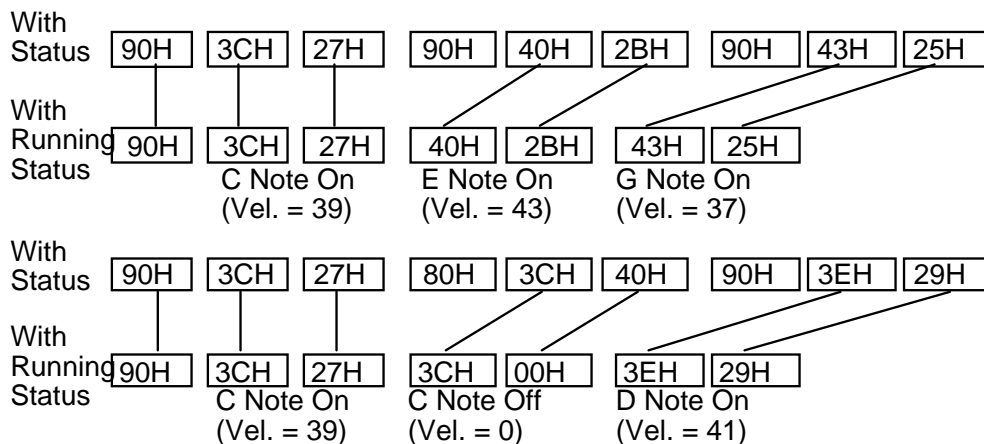
While the above examples pertain to Note On messages, Running Status may also be used for all Mode and Control Change messages. Running Status can drastically reduce the amount of data sent by Continuous Controllers.

In some cases, the receiver must keep the status byte of the mode messages in a Running Status buffer even though the mode message is designated for a channel other than the receiver's basic channel. For example, if an Omni Off mode message is sent followed by Running Status Control Change messages, the Control Change messages can be properly recognized even though the Omni Off message may have been ignored.

Running Status Buffer and Response of Receiver to Different Messages
(Basic Channel = 3, Mode: Omni On)

B0H	7CH	00H

Message:
Omni Off on CH 1 (ignore)

B0H

Running Status
Buffer

01H	37H

Message:
Controller 1 in CH 1
value 37H (recogniz

Valid
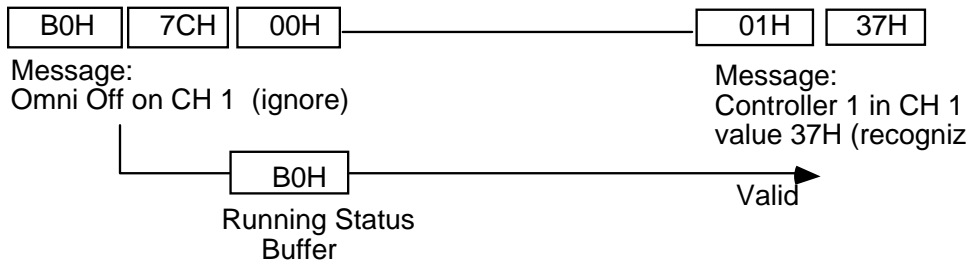
The receiver should always hold the last status byte in a Running Status buffer in case the transmitter is utilizing Running Status to reduce the number of bytes sent. This also means the receiver has to determine how many data bytes (one or two) are associated with each message. It is recommended that the Running Status buffer be set up as follows:

1. Buffer is cleared at power up.
2. Buffer stores the status when a channel message is received.
3. Buffer is cleared when a System Exclusive or Common status message is received.
4. Nothing is done to the buffer during reception of real time messages.
5. The data bytes are ignored when the value of the status buffer is 0 (zero).

There are currently two undefined System Common status bytes (F4H and F5H). Should one of these undefined messages be received, it should be ignored and the running status buffer should be cleared. There are currently two undefined Real Time status bytes (F9H, FDH). Since these may convey only timing information, they should always be ignored, and the running status buffer should remain unaffected.

If Running Status is being used and a receiver is connected to a transmitter after the transmitter has powered on it will not play until the next Status byte is transmitted. For this reason it is recommended that the status be refreshed every few seconds.

To Summarize:

A transmitter may or may not be programmed to take advantage of Running Status. Using Running Status, notes may be turned off by sending a Note On message with zero velocity . It is the responsibility of the receiver to always recognize both normal and running status modes.

A receiver should take into consideration that a transmitter can send messages in either Running Status or normal modes. The following flowchart shows an example of an interrupt-driven routine:

Read Serial Input

Bit 7 = 1   ?   Bit 7 = 0

Yes   ?   Is it a
                        Real-Time
                        Message?
              No
        Store in
      Running Status
         Buffer

        Clear Third
        Byte Flag

No   ?   Is this a
                       Tune Request?

      Yes   = F6H
   (do not   Store it in FIFO
   increment
 pointer here)
                Increment
                Pointer + 1

?   Third Byte Flag = 1

Flag = 0

?   Running Status
           Buffer = 0

Buffer   Ignore
Greater   Data Byte
than 0   Less
            than
      ?   C0H   Clear Third
                                  Byte Flag

Less than E0H   ?   Store Third
                                  Byte into FIFO

                                  Increment
Buffer   Pointer + 3
Greater   ?   Buffer Less
than F0H   than F0H

           ?   Buffer = F2H Clear Running
                                Status Buffer

              Buffer >= F0H

Buffer = F3H   Set Third
or F1H   ?   Byte Flag
Clear Running
Status Buffer   Store Status
                                  into FIFO
Store Status
into FIFO   Store Data Byte
                                  into FIFO
Store Data Byte   Clear Running
into FIFO   Status Buffer (do not
                                  increment
Increment   Ignore Status pointer here)
Pointer + 2

ASSIGNMENT OF NOTE ON/OFF COMMANDS

If an instrument receives two or more Note On messages with the same key number and MIDI channel, it must make a determination of how to handle the additional Note Ons. It is up to the receiver as to whether the same voice or another voice will be sounded, or if the messages will be ignored. The transmitter, however, must send a corresponding Note Off message for every Note On sent. If the transmitter were to send only one Note Off message, and if the receiver in fact assigned the two Note On messages to different voices, then one note would linger. Since there is no harm or negative side effect in sending redundant Note Off messages this is the recommended practice.

VOICE ASSIGNMENT IN POLY MODE

In Poly mode there are no particular rules which define how to assign voices when more than one Note On message is received and recognized. If more Note On messages are transmitted than the receiver is capable of playing, the receiver is free to use any method of dealing with this "overflow" situation (such as first vs. last note priority). The priority of voice assignments may follow the order in which Note On messages are received, the receiver's own keyboard control logic, or some other scheme.

When a transmitter sends Note On and Off information to a slave keyboard which is also being played, it is important for the receiver to distinguish the source of Note On/Off information. For example, a Note Off received from MIDI should not turn off a note that is being played on the slave keyboard. Conversely, releasing a key on the slave's keyboard should not turn off a note being received from MIDI.

"ALL NOTES OFF" FUNCTION WHEN SWITCHING MODES

When a receiver is switching between Omni On/Off and Poly or Mono modes, all notes should be turned off . This is to avoid any unexpected behavior when the instrument's mode is switched. Caution should be taken to turn off only those note events received from MIDI and not those played on the receiver's keyboard.

MIDI MERGING AND ALL NOTES OFF

A sequencer replays previously recorded messages and merges them with any messages received at its MIDI In. A MIDI merging device combines two incoming data streams in real time. In either case the result is that a single MIDI data stream is communicating information produced by more than one transmitter. If an All Notes Off messages is passed through either a sequencer or merging device, then all connected devices will shut off their notes, though the All Notes Off may have only been intended for the notes turned on by one specific instrument. When an All Notes Off is received by a sequencer it should check to make sure that it does not conflict with any notes currently being played. Similarly, if an All Notes Off message is contained in the recorded sequence, it should not be sent if Note On data for that channel is being received. A MIDI merging device should feature the ability to selectively filter All Notes Off messages to avoid this problem.

Mode messages with a second byte greater than 124 should be treated in the same way as the All Notes Off message since they also perform an All Notes Off function.

THE RELATIONSHIP BETWEEN THE HOLD PEDAL AND "ALL NOTES OFF"

If Note Off messages are received while the hold pedal (controller 64 (40H)) is 'on' they must be recognized but not acted upon until the release of the hold pedal. The same is true for the All Notes Off message. A Hold or Sustain pedal 'On' message takes priority over Note Off and All Notes Off until it is released.

All Notes Off should force voices to go to the release stage of the envelope, and not terminate the sound of the notes abruptly.

MIDI Messages Transmitted:

90 43 40	B0 40 7F	90 43 00	B0 7B 00	40 00
(G On)	(HOLD On)	(G Off)	(All Notes Off)	(HOLD Off)

Sound "envelope" of a note being created by the receiving instrument

Notes cannot be cleared here by "All Notes Off", even though Note Off has been received, since hold is still on.
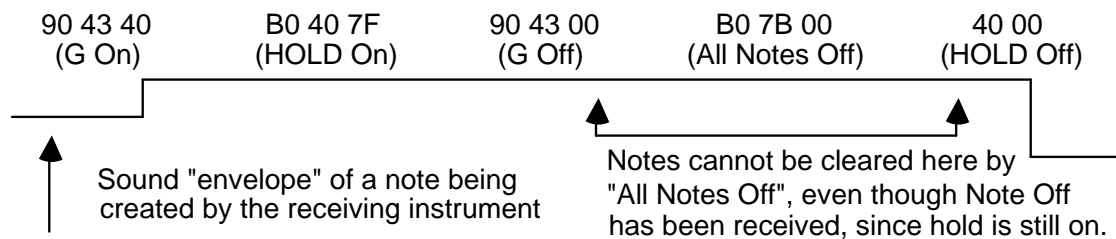
FURTHER DESCRIPTION OF HOLD PEDAL

Hold and Second Release switches use controller number 64 (40H). Proper implementation of the hold pedal will maintain the envelope's sustain level. A "Hold 2" switch has been defined as controller number 69 (45H) as a means of implementing all other hold functions (e.g. "freeze" where envelopes etc. are frozen in their current state) and/or for implementing two different hold functions simultaneously. "Chord Hold" which holds only the notes held when the foot pedal is switched on, is equivalent to the Sostenuto controller 66. All notes played after the foot pedal is switched on are performed normally.

PRIORITY OF MIDI RECEIVING

An instrument capable of receiving and processing incoming MIDI data must give priority to its MIDI In port over its local functions such as the front panel or keyboard. It is critical that incoming data never be ignored or mishandled due to the processor's attention being elsewhere.

At 31.25 Kbaud, one byte is sent every 320 microseconds, which means that proper handling of the received data during any long-term or ongoing MIDI communication will require a high speed microprocessor. For this reason, interrupts and FIFO (first in/first out) buffers are commonly used. As soon as possible after an interrupt is generated, received data can be stored in a FIFO buffer for processing later on. This data handling can take much less than 320 µS, allowing time for the microprocessor to handle other aspects of the instrument's operation.

RELEASE OF OMNI

As a transmitter has no way of knowing what channel a receiver is on it is best to always be able to turn Omni off by means of front panel controls on an instrument.

BASIC CHANNEL OF A SEQUENCER

To a receiver, the output of a MIDI sequencer is identical to the output of any MIDI transmitter with the possible exception of added Real Time bytes. A transmitting instrument sends on a particular channel which a sequencer then records and re-transmits. Thus, a sequencer does not need a Basic Channel as do other instruments. However, this does not prevent a sequencer from having a Basic Channel for recognizing mode messages or changing channels.

TRANSPOSING

If key transpose is implemented on a keyboard instrument, the MIDI key number 60 can be assigned to a physical key other than middle C. Transposition is allowed on both transmitters and receivers. The transposing system in a device should separately affect the keyboard data and incoming MIDI data going to the voice module. To avoid confusion it is a good idea to use an indicator to show when key transpose is active.

MIDI IMPLEMENTATION CHART INSTRUCTIONS

The standard MIDI Implementation Chart is used as a quick reference of transmitter and receiver functions so that users can easily recognize what messages and functions are implemented in the instrument. This chart should be included in the users manual of all MIDI products. For example, if a user intends to connect two MIDI instruments, they might compare the "Transmitted" part of one instrument's chart, with the "Recognized" part of the other instrument's chart by overlapping them. For this reason each chart should be the same size and have the same number of lines.

GENERAL

1. The "[   ]" brackets at the top of the chart is used for the instrument's name such as, [LINEAR WAVETABLE SYNTHESIZER].

2. The item "MODEL" should be used for the model number, such as "LW-1".

3. The body of the implementation chart is divided into four columns. The first column is the specific function or item, the next two columns give information on whether the specified function is transmitted and/or received, and the fourth column is used for remarks. This last column is useful to explain anything unique to this implementation.

FUNCTION DESCRIPTION

1. BASIC CHANNEL:

   Default: Channel which is assigned when power is first applied to unit.
   Changed: The channels which can be assigned from the instrument's front panel.

2. MODE:

   Default: This is the channel mode used by a Transmitter and Receiver when power is
              first applied. This should be written as Mode x (where x is 1 through 4), as shown on
   the   bottom of sheet.
   Messages: These are the mode messages which can be transmitted or received, such as
              OMNI ON/OFF, MONO ON, and POLY ON. MONO ON and POLY ON may be
              written in the short form "MONO", "POLY".
   Altered: This shows the channel modes which are not implemented by a receiver and
              the modes which are substituted. For example, if the receiver cannot accept "MONO
              ON" mode, but will switch to "OMNI ON" mode in order to receive the MIDI data, the
              following expression should be used: "MONO ON > OMNI ON" or "MONO > OMNI".

3. NOTE NUMBER:

   Note Number: The total range of transmitted or recognized notes.
   True Voice: Range of received note numbers falling within the range of true notes
                  produced by the instrument.

4.   VELOCITY:

NOTE ON/NOTE OFF Velocity is assigned either an "o" for implemented or an "x" for not
implemented. In the space following the "o" or "x" it may be mentioned how the
Note On or Off data is being transmitted.

5.   AFTERTOUCH:

"o" for implemented or an "x" for not implemented

6.   PITCH BEND:

"o" for implemented or an "x" for not implemented

7.   CONTROL CHANGE:

Space is given in this area for listing of any implemented control numbers. An "o" or "x" should be
placed in the appropriate Transmitted or Recognized column and the function of the specified
control number should be listed in the remarks column.

8.   PROGRAM CHANGE:

"o" for implemented or an "x" for not implemented. If implemented, the range of numbers should
be included.
True # (Number):   The range of the program change numbers which correspond to the
                          actual number of patches selected.

9.   SYSTEM EXCLUSIVE:

"o" for implemented or an "x" for not implemented. A full description of the instrument's System
Exclusive implementation should be included on separate pages.

10.   SYSTEM COMMON:

"o" for implemented or an "x" for not implemented. The following abbreviations are used:
     Song Pos   = Song Position
     Song Sel   = Song Select
     Tune   = Tune Request

11.   SYSTEM REAL TIME:

"o" for implemented or an "x" for not implemented. The following abbreviations are used:
     Clock   = Timing Clock
     Commands = Start, Continue and Stop

12.   AUX. MESSAGES:

"o" for implemented or an "x" for not implemented. The following abbreviations are used:
     Aux   = Auxiliary
     Active Sense   = Active Sensing

13.   NOTES:
The "Notes" column can be any comments such as:
     Power Up messages transmitted, implementation of program changes to additional
     memory banks, etc.

Model   MIDI Implementation Chart

Function...		Transmitted	Recognized	Remarks
Basic Channel	Default Changed			
Mode	Default Messages Altered	**************		
Note Number	True voice	**************		
Velocity	Note On Note Off			
After Touch	Key's Channel			
Pitch Bend				
Control Change				
Program Change	True Number	**************		
System Exclusive				
System Common	Song Position Song Select Tune Request			
System Real Time	Clock Commands			
Aux Messages	Local On/Off All Notes Off Active Sensing System Reset			
Notes				

Mode 1: Omni On, Poly   Mode 2: Omni On, Mono   O: Yes
Mode 3: Omni Off, Poly   Mode 4: Omni Off, Mono   X: No

TABLE I

SUMMARY OF STATUS BYTES

Hex	STATUS Binary D7--D0		NUMBER OF DATA BYTES	DESCRIPTION
Channel Voice Messages				
8nH	1000nnnn		2	Note Off
9nH	1001nnnn		2	Note On (a velocity of 0 = Note Off)
AnH	1010nnnn		2	Polyphonic key pressure/Aftertouch
BnH	1011nnnn		2	Control change
CnH	1100nnnn		1	Program change
DnH	1101nnnn		1	Channel pressure/After touch
EnH	1110nnnn		2	Pitch bend change
Channel Mode Messages				
BnH	1011nnnn	(01111xxx)	2	Selects Channel Mode
System Messages				
F0H	11110000		*****	System Exclusive
	11110sss		0 to 2	System Common
	11111ttt		0	System Real Time

NOTES:

nnnn:	N-1, where N = Channel #, i.e. 0000 is Channel 1, 0001 is Channel 2, and 1111 is Channel 16.
*****:	0iiiiiii, data, ..., EOX
iiiiiii:	Identification
sss:	1 to 7
ttt:	0 to 7
xxx:	Channel Mode messages are sent under the same Status Byte as the Control Change messages (BnH). They are differentiated by the first data byte which will have a value from 121 to 127 for Channel Mode messages.

TABLE II

CHANNEL VOICE MESSAGES

STATUS		DATA BYTES	DESCRIPTION
Hex	Binary		
8nH	1000nnnn	0kkkkkkk	Note Off
		0vvvvvvv	vvvvvvv: note off velocity
9nH	1001nnnn	0kkkkkkk	Note On
		0vvvvvvv	vvvvvvv ≠ 0: velocity
			vvvvvvv = 0: note off
AnH	1010nnnn	0kkkkkkk	Polyphonic Key Pressure (Aftertouch)
		0vvvvvvv	vvvvvvv: pressure value
BnH	1011nnnn	0ccccccc	Control Change (See Table III)
		0vvvvvvv	ccccccc: control # (0-119)
			vvvvvvv: control value
			ccccccc = 120 thru 127: Reserved. (See Table IV)
CnH	1100nnnn	0ppppppp	Program Change
			ppppppp: program number (0-127)
DnH	1101nnnn	0vvvvvvv	Channel Pressure (Aftertouch)
			vvvvvvv: pressure value
EnH	1110nnnn	0vvvvvvv	Pitch Bend Change LSB
		0vvvvvvv	Pitch Bend Change MSB

NOTES:

1. nnnn: Voice Channel number (1-16, coded as defined in Table I notes)

2. kkkkkkk: note number (0 - 127)

3. vvvvvvv: key velocity
   A logarithmic scale is recommended.

4. Continuous controllers are divided into Most Significant and Least Significant Bytes. If only seven bits of resolution are needed for any particular controllers, only the MSB is sent. It is not necessary to send the LSB. If more resolution is needed, then both are sent, first the MSB, then the LSB. If only the LSB has changed in value, the LSB may be sent without re-sending the MSB.

TABLE III

CONTROLLER NUMBERS

CONTROL NUMBER (2nd Byte value)		CONTROL FUNCTION
Decimal	**Hex**	
0	00H	Bank Select
1	01H	Modulation wheel or lever
2	02H	Breath Controller
3	03H	Undefined
4	04H	Foot controller
5	05H	Portamento time
6	06H	Data entry MSB
7	07H	Channel Volume (formerly Main Volume)
8	08H	Balance
9	09H	Undefined
10	0AH	Pan
11	0BH	Expression Controller
12	0CH	Effect Control 1
13	0DH	Effect Control 2
14-15	0E-0FH	Undefined
16-19	10-13H	General Purpose Controllers (#'s 1-4)
20-31	14-1FH	Undefined
32-63	20-3FH	LSB for values 0-31
64	40H	Damper pedal (sustain)
65	41H	Portamento On/Off
66	42H	Sostenuto
67	43H	Soft pedal
68	44H	Legato Footswitch (vv = 00-3F:Normal, 40-7F=Legatto)
69	45H	Hold 2
70	46H	Sound Controller 1 (default: Sound Variation)
71	47H	Sound Controller 2 (default: Timbre/Harmonic Intensity)
72	48H	Sound Controller 3 (default: Release Time)
73	49H	Sound Controller 4 (default: Attack Time)
74	4AH	Sound Controller 5 (default: Brightness)
75-79	4BH-4FH	Sound Controllers 6-10 (no defaults)
80-83	50-53H	General Purpose Controllers (#'s 5-8)
84	54H	Portamento Control
85-90	55-5AH	Undefined
91	5BH	Effects 1 Depth (formerly External Effects Depth)
92	5CH	Effects 2 Depth (formerly Tremolo Depth)
93	5DH	Effects 3 Depth (formerly Chorus Depth)
94	5EH	Effects 4 Depth (formerly Celeste (Detune) Depth)
95	5FH	Effects 5 Depth (formerly Phaser Depth)
96	60H	Data increment
97	61H	Data decrement
98	62H	Non-Registered Parameter Number LSB
99	63H	Non-Registered Parameter Number MSB
100	64H	Registered Parameter Number LSB
101	65H	Registered Parameter Number MSB
102-119	66-77H	Undefined
120-127	78-7FH	Reserved for Channel Mode Messages

TABLE IIIa

REGISTERED PARAMETER NUMBERS

Parameter Number		Function
LSB	MSB	
00H	00H	Pitch Bend Sensitivity
01H	00H	Fine Tuning
02H	00H	Coarse Tuning
03H	00H	Tuning Program Select
04H	00H	Tuning Bank Select

TABLE IV

CHANNEL MODE MESSAGES

STATUS		DATA BYTES	DESCRIPTION
Hex	Binary		
Bn	1011nnnn	0ccccccc 0vvvvvvv	Mode Messages

ccccccc = 120: All Sound Off
vvvvvvv = 0

ccccccc = 121: Reset All Controllers
vvvvvvv = 0

ccccccc = 122:Local Control
vvvvvvv = 0, Local Control Off
vvvvvvv = 127, Local Control On

ccccccc = 123: All Notes Off
vvvvvvv = 0

ccccccc = 124: Omni Mode Off (All Notes Off)
vvvvvvv = 0

ccccccc = 125:Omni Mode On (All Notes Off)
vvvvvvv = 0

ccccccc = 126: Mono Mode On (Poly Mode Off)
(All Notes Off)
vvvvvvv = M, where M is the number of channels.
vvvvvvv = 0, the number of channels equals the number of voices in the receiver.

ccccccc = 127: Poly Mode On (Mono Mode Off)
(All Notes Off)
vvvvvvv = 0

NOTES:

1. nnnn: Basic Channel number (1-16)

2. ccccccc: Controller number (121 - 127)

3. vvvvvvv: Controller value

TABLE V

SYSTEM COMMON MESSAGES

STATUS		DATA BYTES	DESCRIPTION
Hex	Binary		
F1H	11110001	0nnndddd	MIDI Time Code Quarter Frame nnn: Message Type dddd: Values
F2H	11110010		Song Position Pointer
		01111111 0hhhhhhh	lllllll: (Least significant) hhhhhhh: (Most significant)
F3H	11110011	0sssssss	Song Select sssssss: Song #
F4H	11110100		Undefined
F5H	11110101		Undefined
F6H	11110110	none	Tune Request
F7H	11110111	none	EOX: "End of System Exclusive" flag

TABLE VI

SYSTEM REAL TIME MESSAGES

STATUS		DATA BYTES	DESCRIPTION
Hex	Binary		
F8H	11111000		Timing Clock
F9H	11111001		Undefined
FAH	11111010		Start
FBH	11111011		Continue
FCH	11111100		Stop
FDH	11111101		Undefined
FEH	11111110		Active Sensing
FFH	11111111		System Reset

TABLE VII

SYSTEM EXCLUSIVE MESSAGES

STATUS		DATA BYTES	DESCRIPTION
Hex	Binary		
F0H	11110000		SOX: Start of System Exclusive Status Byte
		0iiiiiii	System Exclusive Sub-ID (see note 1)
			(00 - 7CH) Manufacturer Identification
			(7DH) Non Commercial System Exclusive ID
			(7EH) Non-Real Time System Exclusive
			(7FH) Real Time System Exclusive
		0ddddddd	
		.	Any number of data bytes may be sent here, for any purpose, as
		.	long as they all have a zero in the most significant bit. (see note 2)
		.	
		0ddddddd	
F7H	11110111		EOX: End of System Exclusive

NOTES:

1. 0iiiiiii:

    A) Manufacturer identification (0-124). If the first byte of this ID is 0, the following two bytes are used as extensions to the Manufacturer ID. See Table VIIb for a listing of currently assigned Manufacturer ID numbers. A Manufacturers ID may be obtained from the MIDI Manufacturers Association.
    B) ID 7DH (125) is reserved for non-commercial use (e.g. schools, research, etc.) and is not to be used on any product released to the public.
    C) ID 7EH (126) and 7FH (127) are used for Universal System Exclusive extensions to the MIDI specification. See Table VIIa for a listing of currently defined Non-Real Time and Real Time messages.

2. 0ddddddd:

    All bytes between the System Exclusive Status byte and EOX must have zeroes in the Most Significant Bit -- which therefore makes them Data Bytes -- with the exception of System Real Time Status Bytes (F8H-FFH) (see Table VI). Any other Status Byte that appears between the SOX (F0H) and EOX (F7H) will be considered an EOX message, and terminate the System Exclusive message.

TABLE VIIa

CURRENTLY DEFINED UNIVERSAL SYSTEM EXCLUSIVE MESSAGES

SUB-ID #1	SUB-ID #2	DESCRIPTION
Non-Real Time (7EH)		
00	--	**Unused**
01	(not used)	**Sample Dump Header**
02	(not used)	**Sample Data Packet**
03	(not used)	**Sample Dump Request**
04	nn	**MIDI Time Code**
	00	Special
	01	Punch In Points
	02	Punch Out Points
	03	Delete Punch In Point
	04	Delete Punch Out Point
	05	Event Start Point
	06	Event Stop Point
	07	Event Start Points with additional info.
	08	Event Stop Points with additional info.
	09	Delete Event Start Point
	0A	Delete Event Stop Point
	0B	Cue Points
	0C	Cue Points with additional info.
	0D	Delete Cue Point
	0E	Event Name in additional info.
05	nn	**Sample Dump Extensions**
	01	Multiple Loop Points
	02	Loop Points Request
06	nn	**General Information**
	01	Identity Request
	02	Identity Reply
07	nn	**File Dump**
	01	Header
	02	Data Packet
	03	Request
08	nn	**MIDI Tuning Standard**
	00	Bulk Dump Request
	01	Bulk Dump Reply
09	nn	**General MIDI**
	01	General MIDI System On
	02	General MIDI System Off
7B	(not used)	**End Of File**
7C	(not used)	**Wait**
7D	(not used)	**Cancel**
7E	(not used)	**NAK**
7F	(not used)	**ACK**

Real Time (7FH)

00	--		**Unused**
01	nn		**MIDI Time Code**
	01		Full Message
	02		User Bits
02	nn		**MIDI Show Control**
	00		MSC Extensions
	01 - 7F		MSC Commands
			(Detailed in MSC documentation)
03	nn		**Notation Information**
	01		Bar Number
	02		Time Signature (Immediate)
	42		Time Signature (Delayed)
04	nn		**Device Control**
	01		Master Volume
	02		Master Balance
05	nn		**Real Time MTC Cueing**
	00		Special
	01		Punch In Points
	02		Punch Out Points
	03		(Reserved)
	04		(Reserved)
	05		Event Start points
	06		Event Stop points
	07		Event Start points with additional info.
	08		Event Stop points with additional info.
	09		(Reserved)
	0A		(Reserved)
	0B		Cue points
	0C		Cue points with additional info.
	0D		(Reserved)
	0E		Event Name in additional into.
06	nn		**MIDI Machine Control Commands**
	00 - 7F		MMC Commands
			(Detailed in MMC documentation)
07	nn		**MIDI Machine Control Responses**
	00 - 7F		MMC Commands
			(Detailed in MMC documentation)
08	nn		**MIDI Tuning Standard**
	02		Note Change

NOTES:

1. The standardized format for both Real Time and Non-Real Time messages is as follows:
   F0H <ID number> <device ID> <sub-ID#1> <sub-ID#2>...... F7H

2. Additional details and descriptions of MTC MSC and MMC are available as separate documents.

TABLE VIIb

SYSTEM EXCLUSIVE MANUFACTURER'S ID NUMBERS

NUMBER	MANUFACTURER	NUMBER	MANUFACTURER
American Group			
01H	Sequential	00H 00H 1FH	Zeta Systems
02H	IDP	00H 00H 20H	Axxes
03H	Voyetra/Octave-Plateau	00H 00H 21H	Orban
04H	Moog	00H 00H 24H	KTI
05H	Passport Designs	00H 00H 25H	Breakaway Technologies
06H	Lexicon	00H 00H 26H	CAE
07H	Kurzweil	00H 00H 29H	Rocktron Corp.
08H	Fender	00H 00H 2AH	PianoDisc
09H	Gulbransen	00H 00H 2BH	Cannon Research Group
0AH	AKG Acoustics	00H 00H 2DH	Rogers Instrument Corp.
0BH	Voyce Music	00H 00H 2EH	Blue Sky Logic
0CH	Waveframe Corp	00H 00H 2FH	Encore Electronics
0DH	ADA Signal Processors	00H 00H 30H	Uptown
0EH	Garfield Electronics	00H 00H 31H	Voce
0FH	Ensoniq	00H 00H 32H	CTI Audio, Inc. (Music. Intel Dev.)
10H	Oberheim	00H 00H 33H	S&S Research
11H	Apple Computer	00H 00H 34H	Broderbund Software, Inc.
12H	Grey Matter Response	00H 00H 35H	Allen Organ Co.
13H	Digidesign	00H 00H 37H	Music Quest
14H	Palm Tree Instruments	00H 00H 38H	APHEX
15H	JLCooper Electronics	00H 00H 39H	Gallien Krueger
16H	Lowrey	00H 00H 3AH	IBM
17H	Adams-Smith	00H 00H 3CH	Hotz Instruments Technologies
18H	Emu Systems	00H 00H 3DH	ETA Lighting
19H	Harmony Systems	00H 00H 3EH	NSI Corporation
1AH	ART	00H 00H 3FH	Ad Lib, Inc.
1BH	Baldwin	00H 00H 40H	Richmond Sound Design
1CH	Eventide	00H 00H 41H	Microsoft
1DH	Inventronics	00H 00H 42H	The Software Toolworks
1FH	Clarity	00H 00H 43H	Niche/RJMG
00H 00H 01H	Time Warner Interactive	00H 00H 44H	Intone
00H 00H 07H	Digital Music Corp.	00H 00H 47H	GT Electronics/Groove Tubes
00H 00H 08H	IOTA Systems	00H 00H 4FH	InterMIDI, Inc.
00H 00H 09H	New England Digital	00H 00H 49H	Timeline Vista
00H 00H 0AH	Artisyn	00H 00H 4AH	Mesa Boogie
00H 00H 0BH	IVL Technologies	00H 00H 4CH	Sequoia Development
00H 00H 0CH	Southern Music Systems	00H 00H 4DH	Studio Electronics
00H 00H 0DH	Lake Butler Sound Company	00H 00H 4EH	Euphonix
00H 00H 0EH	Alesis	00H 00H 4FH	InterMIDI
00H 00H 10H	DOD Electronics	00H 00H 50H	MIDI Solutions
00H 00H 11H	Studer-Editech	00H 00H 51H	3DO Company
00H 00H 14H	Perfect Fretworks	00H 00H 52H	Lightwave Research
00H 00H 15H	KAT	00H 00H 53H	Micro-W
00H 00H 16H	Opcode	OOH OOH 54H	Spectral Synthesis
00H 00H 17H	Rane Corp.	OOH OOH 55H	Lone Wolf
00H 00H 18H	Anadi Inc.	00H 00H 56H	Studio Technologies
00H 00H 19H	KMX	00H 00H 57H	Peterson EMP
00H 00H 1AH	Allen & Heath Brenell	00H 00H 58H	Atari
00H 00H 1BH	Peavey Electronics	00H 00H 59H	Marion Systems
00H 00H 1CH	360 Systems	00H 00H 5AH	Design Event
00H 00H 1DH	Spectrum Design and Development	00H 00H 5BH	Winjammer Software
00H 00H 1EH	Marquis Music	00H 00H 5CH	AT&T Bell Labs

NUMBER	MANUFACTURER	NUMBER	MANUFACTURER
00H 00H 5EH	Symetrix	35H	GeneralMusic Corp.
00H 00H 5FH	MIDI the World	39H	Soundcraft Electronics
00H 00H 60H	Desper Products	3BH	Wersi
00H 00H 61H	Micros 'N MIDI	3CH	Avab Electronik Ab
00H 00H 62H	Accordians Intl	3DH	Digigram
00H 00H 63H	EuPhonics	3EH	Waldorf Electronics
00H 00H 64H	Musonix	3FH	Quasimidi
00H 00H 65H	Turtle Beach Systems	00H 20H 00H	Dream
00H 00H 66H	Mackie Designs	00H 20H 01H	Strand Lighting
00H 00H 67H	Compuserve	00H 20H 02H	Amek Systems
00H 00H 68H	BES Technologies	00H 20H 04H	Böhm Electronic
00H 00H 69H	QRS Music Rolls	00H 20H 06H	Trident Audio
00H 00H 6AH	P G Music	00H 20H 07H	Real World Studio
00H 00H 6BH	Sierra Semiconductor	00H 20H 09H	Yes Technology
00H 00H 6CH	EpiGraf Audio Visual	00H 20H 0AH	Audiomatica
00H 00H 6DH	Electronics Deiversified	00H 20H 0BH	Bontempi/Farfisa
00H 00H 6EH	Tune 1000	00H 20H 0CH	F.B.T. Elettronica
00H 00H 6FH	Advanced Micro Devices	00H 20H 0DH	MidiTemp
00H 00H 70H	Mediamation	00H 20H 0EH	LA Audio (Larking Audio)
00H 00H 71H	Sabine Music	00H 20H 0FH	Zero 88 Lighting Limited
00H 00H 72H	Woog Labs	00H 20H 10H	Micon Audio Electronics GmbH
00H 00H 73H	Micropolis	00H 20H 11H	Forefront Technology
00H 00H 74H	Ta Horng Musical Inst.	00H 20H 13H	Kenton Electronics
00H 00H 75H	eTek (formerly Forte)	00H 20H 15H	ADB
00H 00H 76H	Electrovoice	00H 20H 16H	Marshall Products
00H 00H 77H	Midisoft	00H 20H 17H	DDA
00H 00H 78H	Q-Sound Labs	00H 20H 18H	BSS
00H 00H 79H	Westrex	00H 20H 19H	MA Lighting Technology
00H 00H 7AH	NVidia	00H 20H 1AH	Fatar
00H 00H 7BH	ESS Technology	00H 20H 1BH	QSC Audio
00H 00H 7CH	MediaTrix Peripherals	00H 20H 1CH	Artisan Classic Organ
00H 00H 7DH	Brooktree	00H 20H 1DH	Orla Spa
00H 00H 7EH	Otari	00H 20H 1EH	Pinnacle Audio
00H 00H 7FH	Key Electronics	00H 20H 1FH	TC Electonics
00H 00H 80H	Crystalake Multimedia	00H 20H 20H	Doepfer Musikelektronik
00H 00H 81H	Crystal Semiconductor	00H 20H 21H	Creative Technology Pte
00H 00H 82H	Rockwell Semiconductor	00H 20H 22H	Minami/Seiyddo
		00H 20H 23H	Goldstar
European Group		00H 20H 24H	Midisoft s.a.s di M. Cima
20H	Passac	00H 20H 25H	Samick
21H	SIEL	00H 20H 26H	Penny and Giles
22H	Synthaxe	00H 20H 27H	Acorn Computer
24H	Hohner	00H 20H 28H	LSC Electronics
25H	Twister	00H 20H 29H	Novation EMS
26H	Solton	00H 20H 2AH	Samkyung Mechatronics
27H	Jellinghaus MS	00H 20H 2BH	Medeli Electronics
28H	Southworth Music Systems	00H 20H 2CH	Charlie Lab
29H	PPG	00H 20H 2DH	Blue Chip Music Tech
2AH	JEN	00H 20H 2EH	BEE OH Corp
2BH	SSL Limited		
2CH	Audio Veritrieb		
2FH	Elka		
30H	Dynacord		
31H	Viscount		
33H	Clavia Digital Instruments		
34H	Audio Architecture		

NUMBER	MANUFACTURER	NUMBER	MANUFACTURER

Japanese Group (as of 10/92)

40H	Kawai
41H	Roland
42H	Korg
43H	Yamaha
44H	Casio
46H	Kamiya Studio
47H	Akai
48H	Japan Victor
49H	Mesosha
4AH	Hoshino Gakki
4BH	Fujitsu Elect
4CH	Sony
4DH	Nisshin Onpa
4EH	TEAC
50H	Matsushita Electric
51H	Fostex
52H	Zoom
53H	Midori Electronics
54H	Matsushita Communication Industrial
55H	Suzuki Musical Inst. Mfg.

TABLE VIII

ADDITIONAL OFFICIAL SPECIFICATION DOCUMENTS PUBLISHED BY
THE MIDI MANUFACTURERS ASSCOCIATION

DOCUMENT TITLE	DESCRIPTION
MIDI Time Code	Recommended Practice RP004/RP008
MIDI Show Control 1.1	Recommended Practice RP002 /RP014*
MIDI Machine Control	Recommended Practice RP013
Standard MIDI Files	Recommended Practice RP001
General MIDI System Level 1	Recommended Practice RP003

*New Version, February 1996

Note: All these specifications are included in this MMA "Complete Detailed MIDI 1.0 Specification" book.

MIDI Time Code

MIDI Time Code

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

Quarter Frame Messages

Quarter Frame messages are used only while the system is running. They are rather like the PPQN or MIDI clocks to which we are accustomed. But there are several important ways in which Quarter Frame messages differ from the other systems.

As their name implies, they have fine resolution. If we assume 30 frames per second, there will be 120 Quarter Frame messages per second. This corresponds to a maximum latency of 8.3 milliseconds (at 30 frames per second), with accuracy greater than this possible within the specific device (which may interpolate in between quarter frames to "bit" resolution). Quarter Frame messages serve a dual purpose: besides providing the basic timing pulse for the system, each message contains a unique nibble (four bits) defining a digit of a specific field of the current SMPTE time.

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

Quarter Frame Messages (2 bytes):

```
F1   <message>
```

`F1`	System Common status byte
`<message>`	`0nnn dddd`
`nnn`	Message Type:
	0 = Frame count LS nibble
	1 = Frame count MS nibble
	2 = Seconds count LS nibble
	3 = Seconds count MS nibble
	4 = Minutes count LS nibble
	5 = Minutes count MS nibble
	6 = Hours count LS nibble
	7 = Hours count MS nibble and SMPTE Type
`dddd`	4 bits of binary data for this Message Type

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

FRAME COUNT: xxx yyyyy

xxx   Undefined and reserved for future use. Transmitter must set these bits to 0 and receiver should ignore!

yyyyy   Frame count (0-29)

SECONDS COUNT: xx yyyyyy

xx   Undefined and reserved for future use. Transmitter must set these bits to 0 and receiver should ignore!

yyyyyy   Seconds Count (0-59)

MINUTES COUNT: xx yyyyyy

xx   Undefined and reserved for future use. Transmitter must set these bits to 0 and receiver should ignore!

yyyyyy   Minutes Count (0-59)

HOURS COUNT: x yy zzzzz

x   Undefined and reserved for future use. Transmitter must set this bit to 0 and receiver should ignore!

yy   Time Code Type:
               0 = 24 Frames/Second
               1 = 25 Frames/Second
               2 = 30 Frames/Second (Drop-Frame)
               3 = 30 Frames/Second (Non-Drop)

zzzzz   Hours Count (0-23)

Quarter Frame Message Implementation

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

```
F1 0X
F1 1X
F1 2X
F1 3X
F1 4X
F1 5X
F1 6X
F1 7X
```

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

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

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

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

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

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

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

and know exactly where it is in relation to the 2-frame sequence, down to a quarter frame accuracy.

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

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

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

```
F1 00
F1 11          10H = 16 decimal

F1 24
F1 33          34H = 52 decimal

F1 45
F1 52          25H = 37 decimal

F1 61
F1 76          01H = 01 decimal (SMPTE Type is 30 frames/non-drop)
```

(note: the value transmitted is "6" because the SMPTE Type (11 binary) is encoded in bits 5 and 6)

For SMPTE Types of 24, 30 drop frame, and 30 non-drop frame, the frame number will always be even. For SMPTE Type of 25, the frame number may be even or odd, depending on which frame number the 8-message sequence had started. In this case, you can see where the MIDI Time Code frame number would alternate between even and odd every second.

MIDI Time Code will take a very small percentage of the MIDI bandwidth. The fastest SMPTE time rate is 30 frames per second. The specification is to send 4 messages per frame - in other words, a 2-byte message (640 microseconds) every 8.333 milliseconds. This takes 7.68 % of the MIDI bandwidth - a reasonably small amount. Also, in the typical MIDI Time Code systems we have imagined, it would be rare that normal MIDI and MIDI Time Code would share the same MIDI bus at the same time.

NOTE: When a VITC signal drives a MIDI Time Code system through a SMPTE-to-MIDI converter, the MIDI Time Code frames do not advance by two as expected. They may advance by one or not at all, and time code which was even frame numbers may become odd frame numbers. To accomplish synchronization, it necessary to wait until the first four (at least) bits of the SMPTE have been received before sending the MTC so you know if the frame has advanced or not.

Full Message

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

Full Message - (10 bytes)

```
F0 7F <device ID> 01 <sub-ID 2> hr mn sc fr F7
```

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

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

User Bits

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

User Bits Message - (15 bytes)

```
F0 7F <device ID> 01 <sub-ID 2> u1 u2 u3 u4 u5 u6 u7 u8 u9 F7
```

`F0 7F`	Real Time Universal System Exclusive Header
`<device ID>`	7F (message intended for entire system)
`01`	`<sub-ID 1>` = MIDI Time Code
`<sub-id 2>`	02, User Bits Message
`u1`	`0000aaaa`
`u2`	`0000bbbb`
`u3`	`0000cccc`
`u4`	`0000dddd`
`u5`	`0000eeee`
`u6`	`0000ffff`
`u7`	`0000gggg`
`u8`	`0000hhhh`
`u9`	`000000ji`
`F7`	EOX

Message bytes `u1` through `u8` correspond to SMPTE/EBU Binary Groups 1 through 8, respectively. Byte `u9` contains the SMPTE/EBU Binary Group Flag Bits, where `j` corresponds to SMPTE time code bit 59 (EBU bit 43), and `i` corresponds to SMPTE time code bit 43 (EBU bit 27).

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: `hhhhgggg ffffeeee ddddcccc bbbbaaaa`. * If they are used to carry time code number in BCD form (a common practice), then the frames units would go into Group 1, and the hours tens in Group 8. To display correctly, on would start with Group 8 and finish with Group 1 - again, `hhhhgggg ffffeeee ddddcccc bbbbaaaa`.

This message can be sent whenever the User Bits values must be transferred to any devices down the line. Note that the User Bits Message may be sent by the MIDI Time Code Converter at any time. It is not sensitive to any mode.

> *Note: This message was redefined in November 1991 to more accurately reflect the way SMPTE time code is read. The original version specified that the "nibble fields decode in an 8-bit format: `aaaabbbb ccccdddd eeeeffff gggghhhh ii`".

MIDI Cueing

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

Of 128 possible event types, the following are currently defined:

MIDI Cueing Set-Up Messages (13 bytes plus any additional information):

```
F0 7E <device ID> 04 <sub-ID 2> hr mn sc fr ff sl sm <add. info.> F7
```

F0 7E	Non-Real Time Universal System Exclusive Header
<device ID>	Device number of unit
04	<sub-ID 1> = MIDI Time Code
<sub-ID 2>	Set-Up Type
00	Special
01	Punch In points
02	Punch Out points
03	Delete Punch In point
04	Delete Punch Out point
05	Event Start points
06	Event Stop points
07	Event Start points with additional info.
08	Event Stop points with additional info.
09	Delete Event Start point
0A	Delete Event Stop point
0B	Cue points
0C	Cue points with additional info.
0D	Delete Cue point
0E	Event Name in additional info.
hr	hours and type: 0 yy zzzzz
yy	type:
	00 = 24 Frames/Second
	01 = 25 Frames/Second
	10 = 30 Frames/Second drop frame
	11 = 30 Frames/Second non-drop frame
zzzzz	Hours (00-23)
mn	Minutes (00-59)
sc	Seconds (00-59)
fr	Frames (00-29)
ff	Fractional Frames (00-99)
sl, sm	Event Number (LSB first)
<add. info.>	
F7	EOX

Description Of MTC Cueing Set-Up Types

`00`   **Special** refers to the set-up information that affects a unit globally (as opposed to individual tracks, sounds, programs, sequences, etc.). In this case, the **Special Type** takes the place of the Event Number. Five are defined. Note that types `01 00` through `04 00` ignore the event time field.

        `00 00` **Time Code Offset** refers to a relative Time Code offset for each unit. For example, a piece of video and a piece of music that are supposed to go together may be created at different times, and more than likely have different absolute time code positions - therefore, one must be offset from the other so that they will match up. Just like there is one master time code for an entire system, each unit only needs one offset value per unit.

        `01 00` **Enable Event List** means for a unit to enable execution of events in its list if the appropriate MTC or SMPTE time occurs.

        `02 00` **Disable Event List** means for a unit to disable execution of its event list but not to erase it. This facilitates an MTC Event Manager in muting particular devices in order to concentrate on others in a complex system where many events occur simultaneously.

        `03 00` **Clear Event List** means for a unit to erase its entire event list.

        `04 00` **System Stop** refers to a time when the unit may shut down. This serves as a protection against Event Starts without matching Event Stops, tape machines running past the end of the reel, and so on.

        `05 00` **Event List Request** is sent by a master to an MTC peripheral. If the device ID (Channel Number) matches that of the peripheral, the peripheral responds by transmitting its entire cue list as a sequence of Set Up Messages, starting from the SMPTE time indicated in the Event List Request message.

`01/02` **Punch In** and **Punch Out** refer to the enabling and disabling of record mode on a unit. The Event Number refers to the track to be recorded. Multiple punch in/punch out points (and any of the other event types below) may be specified by sending multiple Set-Up messages with different times.

`03/04` **Delete Punch In or Out** deletes the matching point (time and event number) from the Cue List.

`05/06` **Event Start and Stop** refer to the running or playback of an event, and imply that a large sequence of events or a continuous event is to be started or stopped. The event number refers to which event on the targeted slave is to be played. A single event (i.e. playback of a specific sample, a fader movement on an automated console, etc.) may occur several times throughout a given list of cues. These events will be represented by the same event **number**, with different Start and Stop times.

`07/08` **Event Start and Stop with Additional Information** refer to an event (as above) with additional parameters transmitted in the Set Up message between the Time and EOX. The additional parameters may take the form of an effects unit's internal parameters, the volume level of a sound effect, etc. See below for a description of additional information.

`09/0A` **Delete Event Start/Stop** means to delete the matching (event number and time) event (with or without additional information) from the Cue List.

`0B` **Cue Point** refers to individual event occurrences, such as marking "hit" points for sound effects, reference points for editing, and so on. Each Cue number may be assigned to a specific reaction, such as a specific one-shot sound event (as opposed to a continuous event, which is handled by Start/Stop). A single cue may occur several times throughout a given list of cues. These events will be represented by the same event **number**, with different Start and Stop times.

`0C` **Cue Point with Additional Information** is exactly like Event Start/Stop with Additional Information, except that the event represents a Cue Point rather than a Start/Stop Point.

`0D` **Delete Cue Point** means to Delete the matching (event number and time) Cue Event with or without additional information from the Cue List.

`0E` **Event Name in Additional Information**. This merely assigns a name to a given event number. It is for human logging purposes. See Additional Information description.

EVENT TIME: This is the SMPTE/MIDI Time Code time at which the given event is supposed to occur. Actual time is in 1/100th frame resolution, for those units capable of handling bits or some other form of sub-frame resolution, and should otherwise be self-explanatory.

EVENT NUMBER: This is a fourteen-bit value, enabling 16,384 of each of the above types to be individually addressed. "sl" is the 7 LS bits, and "sm" is the 7 MS bits.

ADDITIONAL   Additional information consists of a nibblized MIDI data stream,
INFORMATION: LS nibble first. The exception is Set-Up Type `0E`, where the additional information is nibblized ASCII, LS nibble first. An ASCII newline is accomplished by sending CR and LF in the ASCII. CR alone functions solely as a carriage return, and LF alone functions solely as a Line-Feed.

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

Real Time MIDI Cueing Set-Up Messages (8 bytes plus any additional information):

```
F0 7F <device ID> 05 <sub-id #2> sl sm <additional info.> F7
```

`F0 7F`	Universal Real Time SysEx Header	
`<device ID>`	ID of target device	
`05`	`<sub id #1>` = MIDI Time Code Cueing Messages	
`<sub-ID #2>`	Set-Up Type	
`00`	Special	
	Event Number = `04 00` = System Stop (All others reserved)	
`01`	Punch In points	
`02`	Punch Out points	
`03`	(Reserved)	
`04`	(Reserved)	
`05`	Event Start points	
`06`	Event Stop points	
`07`	Event Start points with additional info.	
`08`	Event Stop points with additional info.	
`09`	(Reserved)	
`0A`	(Reserved)	
`0B`	Cue points	
`0C`	Cue points with additional info.	
`0D`	(Reserved)	
`0E`	Event Name in additional info.	
`sl, sm`	Event Number (LSB first)	
`<add. info.>`	nibblized as per the MTC Cueing Specification	
`F7`	EOX	

Real Time MTC Cueing essentially duplicates the bulk of the Non-Real time MTC Cueing messages in the Universal Real-Time area.

Note that the time field has been dropped. With this message the time would be "as soon as you receive this." All of the Delete messages plus many of the Special messages have been excluded – they were intended for remote-editing of a cue list and are not needed for real-time response.

The format and definitions of all other messages remains the same, as do their sub ID #2 definitions (in order to match one-on-one with their Non-Real Time counterparts).

Refer to the Non-Real Time MTC definitions (above) for more detailed information.

Potential Problems

There is a possible problem with MIDI mergers created before MTC was defined improperly handling the `F1` message, since they will not know how many bytes are following. However, in typical MIDI Time Code systems, we do not anticipate applications where the MIDI Time Code must be merged with other MIDI signals occurring at the same time.

Please note that there is plenty of room for additional set-up types, etc., to cover unanticipated situations and configurations.

It is recommended that each MTC peripheral power up with its MIDI Manufacturer's System Exclusive ID number as its default device ID. Obviously, it would be preferable to allow the user to change this number from the device's front panel, so that several peripherals from the same manufacturer may have unique IDs within the same MTC system.

In most cases, the device ID acts just like a channel number — this is how you address individual pieces of equipment with universal system exclusive messages. Thus, the channel number works as a good default. However, in large systems, a master controller or computer may wish to individually address different instruments that are on the same MIDI channel — therefore, the device ID should be user-adjustable to sort out such a problem. It is also acceptable for a device to power up to the state it was in when last powered down.

MTC Signal Path Summary

Data sent between the Master Time Code Source (which may be, for example, a Multitrack Tape Deck with a SMPTE Synchronizer) and the MIDI Time Code Converter is <u>always SMPTE Time Code</u>.

Data sent from the MIDI Time Code Converter to the Master Control/Cue Sheet (note that this may be a MTC-equipped tape deck or mixing console as well as a cue-sheet) is <u>always MIDI Time Code</u>. The specific MIDI Time Code messages which are used depend on the current operating mode, as explained below:

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

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

Because the tape is being moved by hand in Cue Mode, the tape direction can change quickly and often. The order of the Frame Message sequence must change along with the tape direction.

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

SHUTTLE MODE: This is just another expression for "Fast-Forward/Rewind Mode".

Reference

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

Standard MIDI Files 1.0

RP001

Revised February 1996

Copyright © 1988, 1994, 1996 MIDI Manufacturers Association

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 used 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 <u>variable-length quantity</u>. 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 0FFFFFFF 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 x 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 <u>chunks</u>. 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 <u>header</u> chunk provides a minimal amount of information pertaining to the entire MIDI file. A <u>track</u> 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
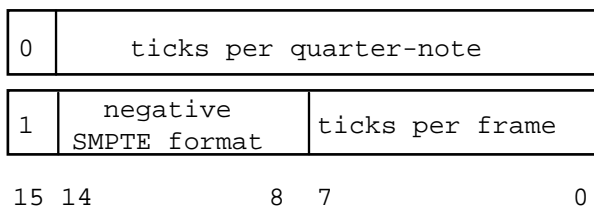
If bit 15 of `<division>` is a zero, the bits 14 thru 0 represent the number of delta-time "ticks" which make up a quarter-note. For instance, if `<division>` is 96, then a time interval of an eighth-note between two events in the file would be 48.

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 an 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.

Standard MIDI Files 1.0

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                              200-tick delta-time
F7 06 43 12 00 43 12 00
64                                 100-tick delta-time
F7 04 43 12 00 F7
```

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 *not* 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 *must* 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`.

```c
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):

<pre>
                    4D 54 72 6B   MTrk
                    00 00 00 3B   chunk length (59)

Delta-time   Event   Comments
   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
</pre>

A format 1 representation of the file is slightly different.

First, its header chunk:

<pre>
                    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
</pre>

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

<pre>
                    4D 54 72 6B   MTrk
                    00 00 00 14   chunk length (20)

Delta-time   Event   Comments
   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
</pre>

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

<pre>
                    4D 54 72 6B   MTrk
                    00 00 00 10   chunk length (16)

Delta-time   Event   Comments
   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
</pre>

Then, the track chunk for the second music track:

```
4D 54 72 6B              MTrk
00 00 00 0F              chunk length (15)
```

Delta-time	Event	Comments
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              MTrk
00 00 00 15              chunk length (21)
```

Delta-time	Event	Comments
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:

Time (in ms.) = (Number of Ticks) * (Tempo (uS/qn) / 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:

Time = 6144 * (500000/96) / 1000
    = 32000 milliseconds

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

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

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.

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:

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

Registered Parameter #	Description
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

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 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.

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

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 hhhhh
        tt  = time type (bit format):
                00 = 24 frame
                01 = 25 frame
                10 = 30 drop frame
                11 = 30 frame
        hhhhh = 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  bbbbbbb (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_2O
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	MTC_CHASE_ON	variable	3
1A	MTC_CHASE_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 MTC_CHASE_ON

```
19                      MTC_CHASE_ON
<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 MTC_CHASE_OFF

```
1A                      MTC_CHASE_OFF
<Q_list>                optional
```

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

MTC_CHASE_OFF 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. Manufactures 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 does 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 my 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:

$$s1 = (\text{status\_code} / 4) \; \& \; 7F$$
$$s2 = (\text{status\_code} / 512) \; \& \; 7F$$
$$\text{status\_code} = (s1 * 4) + (s2 * 512)$$

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.

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

For command_format values between 01 and 0F (lighting)

hex	messages	description
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

For command_format values between 10 and 1F (sound)

hex	messages	description
10 04	S G	amplifier failure
10 08	S G	amplifier overload

For command_format values between 20 and 2F (machinery)

hex	messages	description
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

For command_format values between 30 and 3F (video)

hex	messages	description
10 04	S G	sync lost
10 08	S G	time code lost

For command_format values between 40 and 4F (projection)

hex	messages	description
10 04	S G	film tension lost
10 08	S G	lamp failure

For command_format values between 50 and 5F (process control)

hex	messages	description
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

For command_format values between 60 and 6F (pyrotechnics)

hex	messages	description
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 to 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 <command_format>=24 and <Q_number>=34.1 and L-119 would have <command_format>=01 and <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

MIDI Machine Control Recommended Practice 1.00

January, 1992

CONTENTS

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 ESbus standard. The intention is that command translation between the MIDI Machine Control specification and the ESbus 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 ESbus standard.

By contrast with ESbus 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:

```
   _____
  |                                   |
  | CLOSED_LOOP_STATE                 |
  | (default at power up)             |
  |                                   |  <<<-----
  | Request response(s) from          |          |
  | the controlled device             |          |
  |_____|          |
                 |                                |
  =========================       =====================
  Response time-out (2 sec)       Arrival of a response
  =========================       =====================
                 |                _____|_____
                 |               |                        |
                 |               | OPEN_LOOP_STATE        |
                 |               |                        |
       -------->>> | If the Controller wishes to  |
                   | detect further changes in    |
                   | the state of the loop, then  |
                   | it should transmit a request |
                   | for a response (any response)|
                   | at some regular interval.    |
                   | Subsequent time-outs, while  |
                   | waiting for responses to     |
                   | arrive, should be            |
                   | transparent to the operation |
                   | of the Controller.           |
                   |_____|
```

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:	From Controlled Device to Controller:
`F0 7F <all_call=7F> <mcc> <WAIT> F7`	`F0 7F <device_ID> <mcr> <WAIT> F7`
`F0 7F <all_call=7F> <mcc> <RESUME> 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.

4

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 <u>be able to determine the length of any received message, whether that message is known to the device or not</u>.
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 hhhhh
                        tt = time type:
                                00 = 24 frame
                                01 = 25 frame
                                10 = 30 drop frame
                                11 = 30 frame
                        hhhhh = hours (0-23 decimal, encoded as 00-17 hex)
                mn = Minutes: 0 c mmmmmm
                        c = color frame flag (copied from bit in time code stream):
                                0 = non color frame
                                1 = color framed code
                        mmmmmm = 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.
                        ssssss = seconds (0-59 decimal, encoded as 00-3B hex)
                fr = Frames, byte 5 ident and sign: 0 g i fffff
                        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 bbbbbbb (0-99 decimal, encoded as 00-63 hex)

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

> st = code status: $0\ e\ v\ d\ n\ xxx$
>> 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 <u>never</u> been read since power up or an MMC RESET
>>
>> xxx = 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 *hhhhgggg ffffeeee ddddcccc bbbbaaaa*.
4. Display order for the userbits digits will also be *hhhhgggg ffffeeee ddddcccc bbbbaaaa*.

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 $qqqqqqq$
sl = LSB of nominal fractional part of speed value: 0 $rrrrrrr$

Speed values per shift left count:

| | BINARY REPRESENTATION | | USEABLE RANGES (DECIMAL) | |
| | Integer multiple | Fractional part | Integer | Fractional |
sss	of play-speed	of play-speed	range	resolution
000	ppp .	$qqqqqqqrrrrrrr$	0-7	$1/16384$
001	$pppq$.	$qqqqqqrrrrrrr$	0-15	$1/8192$
010	$pppqq$.	$qqqqqrrrrrrr$	0-31	$1/4096$
011	$pppqqq$.	$qqqqrrrrrrr$	0-63	$1/2048$
100	$pppqqqq$.	$qqqrrrrrrr$	0-127	$1/1024$
101	$pppqqqqq$.	$qqrrrrrrr$	0-255	$1/512$
110	$pppqqqqqq$.	$qrrrrrrr$	0-511	$1/256$
111	$pppqqqqqqq$.	$rrrrrrr$	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,
	(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 it's 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

13

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 (Evnt)
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 <u>exit from record [rehearse]</u>.

> *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 <u>do not automatically exit from record [rehearse]</u> 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

21

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:
REQUESTED OFFSET = SELECTED TIME CODE - 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)
.	
.	

43 UPDATE

FORMAT 1 - UPDATE [BEGIN]

UPDATE [BEGIN] causes a Controlled Device to <u>immediately</u>:
    (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, <u>at intervals no more frequent than that specified by the UPDATE RATE Information Field</u>,
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
<u>always</u> be in the full <u>Standard Time Code</u> (5-byte) format, while subsequent responses will use the
<u>Standard Short Time Code</u> (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                    LOCATE [TARGET]
<count=06>            Byte count.
01                    "TARGET" sub-command
hr mn sc fr ff        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                    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 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:

tt (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.
c (color frame)	*0*
k (blank)	*0*
g (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 (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:

tt, c, k, g	Same as *{ff}* field
i (final byte id)	*1*
e (estimated code)	*0*
v (invalid code)	*0*
d (video field 1)	*0/1* as determined
n (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:

[Destination] = [Source #1] - [Source #2]

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

4E	SUBTRACT
<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.

4F   DROP FRAME ADJUST

Convert the contents of the named Information Field into <u>drop-frame</u> 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	DROP FRAME ADJUST
<count=01+ext>	Byte count (not including command and count).
<name>	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	PROCEDURE [ASSEMBLE]
<count=variable>	Byte count.
00	"ASSEMBLE" sub-command.
<procedure>	Procedure Name in the range 00 thru 7E.
	7F is reserved.
<command #1..>	Any MMC commands except:
<command #2..>	(i) another PROCEDURE [ASSEMBLE]; *2
<command #3..>	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 delete all PROCEDURE's.

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 set all PROCEDURE's.

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 = <u>Delete Event definition immediately upon being triggered (ESbus mode),</u>
	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

$$0D = GP5$$
$$0E = GP6$$
$$0F = GP7$$

<command..>   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 <u>when the Event is defined</u>, 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 delete all EVENT's.

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 set all EVENT's.

FORMAT 4 - EVENT [TEST]

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

38

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 unused 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 all 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 all 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 <u>do not automatically exit from record [rehearse]</u> 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 (<u>i.e. the "Most recently activated Motion Control State" in the MOTION CONTROL TALLY Information Field is PLAY or VARIABLE PLAY</u>), 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 (<u>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"</u>), 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	RECORD STROBE VARIABLE
<count=03>	Byte count.
sh sm sl	Standard Speed Specification

NOTES:

1. <u>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.</u> 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 <u>only under the STOP or PAUSE conditions specified</u>. 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:
   (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 command is always the only command in its Sysex, and is directed to the "all-call" address i.e. *FO 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. *FO 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
hr mn sc fr st	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
hr mn sc fr st	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 <u>non-drop-frame number</u> 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, <u>at least one General Purpose register must be supported</u> 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.

<name>	Short time code Information Field name
fr {st|ff}	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.

40	SIGNATURE
<count=variable>	Byte count of all subsequent data. *1
vi	MMC Version implemented by the device; integer part, converted to binary. For the current version *vi=01*.
vf	MMC Version implemented by the device; fractional part, *00* thru *99*, converted to binary (*00-63h*). For current version, *vf=00*.
va	reserved, must be set to *00* (MMC version extension)
vb	reserved, must be set to *00* (MMC version extension)
<count_1>	Byte count for <u>Command Bitmap Array</u>
c0	Command bitmap 0: Commands 00 thru 06: *0 gfedcba*
	a = Command 00
	b = Command 01
	c = Command 02
	d = Command 03
	e = Command 04
	f = Command 05
	g = Command 06
c1	Command bitmap 01: Commands 07 thru 0D
c2	Command bitmap 02: Commands 0E thru 14
c3	Command bitmap 03: Commands 15 thru 1B
c4	Command bitmap 18: Commands 1C thru 1F: *0000 dcba*
	a = command 1C
	b = command 1D
	c = Command 1E
	d = Command 1F
c5	Command bitmap 05: Commands 20 thru 26
c6	Command bitmap 06: Commands 27 thru 2D
c7	Command bitmap 07: Commands 2E thru 34
c8	Command bitmap 08: Commands 35 thru 3B
c9	Command bitmap 09: Commands 3C thru 3F

c10	Command bitmap 10: Commands 40 thru 46
c11	Command bitmap 11: Commands 47 thru 4D
c12	Command bitmap 12: Commands 4E thru 54
c13	Command bitmap 13: Commands 55 thru 5B
c14	Command bitmap 14: Commands 5C thru 5F
c15	Command bitmap 15: Commands 60 thru 66
c16	Command bitmap 16: Commands 67 thru 6D
c17	Command bitmap 17: Commands 6E thru 74
c18	Command bitmap 17: Commands 75 thru 7B
c19	Command bitmap 17: Commands 7C thru 7F
c20 thru c39	Command bitmaps 20 thru 39: Extended commands 00 01 thru 00 7F
c40 thru c59	Command bitmaps 40 thru 59: Extended commands 00 00 01 thru 00 00 7F
\<count_2\>	Byte count for <u>Response/Information Field Array</u>
r0 thru r19	Response/Information Field bitmaps 00 thru 19: Responses and Information Fields 00 thru 7F
r20 thru r39	Response/Information Field bitmaps 20 thru 39: Extended Responses and Information Fields 00 01 thru 00 7F
r40 thru r59	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 only 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 <u>less than or equal to</u> the COMMAND ERROR LEVEL value, then that error is enabled. If <u>greater than</u>, 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.

<count_1>	Byte count of following offset and command string. (For "Receive buffer overflow" and "Command Sysex length" errors, or if ee = 7F, set <count_1> = 00 and omit <offset> and <command string>.)
<offset>	Offset relative to the start of <command string> of the byte which caused the error (<offset=00> for first byte of string). Set to 7F if byte position is unavailable or undefined due to the nature of the error.
<command string>	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> <count> <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)

        <u>Reaction to a "disabled" MAJOR ERROR is as follows:</u>
      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 <u>less than or equal to</u> the COMMAND ERROR LEVEL value, then that error is enabled. If <u>greater than</u>, the error is disabled.
The default condition, after power up or an MMC RESET, is "All errors disabled".

44	COMMAND ERROR LEVEL
`<count=01>`	Byte count.
vv	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	TIME STANDARD
<count=01>	Byte count.
<type>	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	SELECTED TIME CODE SOURCE .
<count=01>	Byte count.
ss	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 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".

57

*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	VELOCITY TALLY
<count=03>	Byte count.
sh sm sl	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	STOP MODE
<count=01>	Byte count.
cc	Mode code:
	00 = Disable monitoring.
	01 = Enable monitoring.*1
	7F = 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 <u>not</u> 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

61

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	RECORD MONITOR
<count=01>	Byte count.
dd	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 = *32*hex (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 it's 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

5B	GENERATOR COMMAND TALLY
<count=02>	Byte count.
gg	Most recent generator command:
	00 = Stop
	01 = Run
	02 = Copy/Jam
ss	Status (bit = 1 if status true) and success level: *0 0 cb 0 aaa*
	aaa = Success level:
	000 = Transition in progress
	001 = Successful
	010 = Failure
	b = Loss of Time Code Source data (during Copy/Jam)
	c = Loss of External Frame Sync Reference

5C   GENERATOR SET UP [read/write]

Controls the operating modes of the time code generator.

5C	GENERATOR SET UP
<count=03+ext>	Byte count (not including command and count).
<reference>	Generator Frame Sync References: *0 yyy 0 nnn*
	nnn = Frame Sync Reference for Run mode:
	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]
	yyy = Frame Sync Reference for Copy/Jam mode:
	000 = Time Code Source frame edges
	001 = Locally defined external Frame Reference
	111 = As locally defined [write only]
<source>	Time Code Source for Copy/Jam:
	00 = reserved for extensions
	01 = SELECTED TIME CODE (Default)
	02 = SELECTED MASTER CODE
	7F = As locally defined [write only]
<copy/jam>	Copy/Jam mode:
	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</u> mode).

NOTES:

*1   Internal Crystal Rate Table:

	GENERATOR TIME CODE time type (tt)			
nnn	24	25	30DF	30
000 (Standard)	24	25	29.97	30
010 (Drop A)	24	25	29.97	29.97
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 <u>not</u> 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).
hr mn sc fr ff	Event time (type *{ff}*).
<command..>	Single command plus data.

62   TRACK MUTE [read/write]

Selects individual tracks that will have their Output signals muted.

<u>TRACK MUTE overrides all other monitoring selections</u>.

<u>When read</u> 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 <u>not</u> to be selected for individual muting. A Byte count of 00 may be used if no tracks are muted.
<u>When written to</u> 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;
	0Ahex 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*.
	0Chex 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
	ssssss = 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
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 *01* hex.
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>
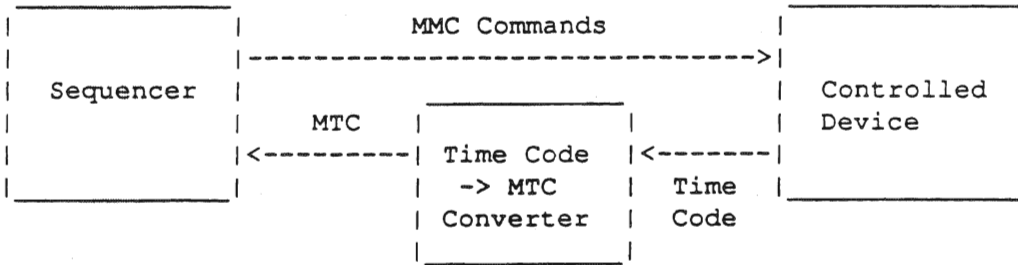*F7*

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:

```
 _____                  MMC Commands                    _____
|               |     |------------------------------------->|  |               |
|               |     |                                      |  |               |
|   Sequencer   |     |           _____             |  |  Controlled   |
|               |     |   MTC    |              |             |  |  Device       |
|               |     |<---------| Time Code    |  |<-------| |  |               |
|               |     |          | -> MTC       |  | Time   | |  |               |
|_____|     |          | Converter    |  | Code   | |  |_____|
                                 |_____|
```

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 *01*hex. 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

76

Operator clicks "Remote Machine PLAY" to punch out of record:

F0 7F 01 <mcc> <RECORD EXIT> <DEFERRED PLAY> F7

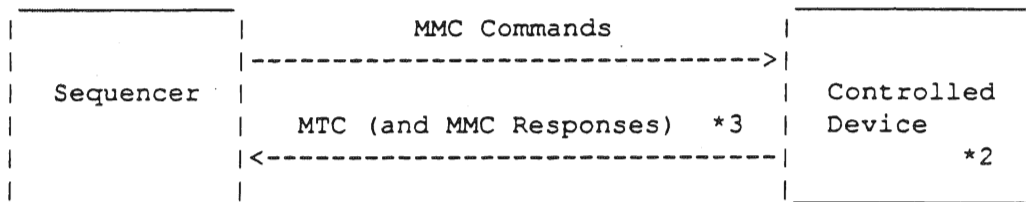Operator clicks "Remote Machine STOP":

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:

```
 _____                MMC Commands            _____
|               |  |                                  | |               |
|               |  |--------------------------------->| |               |
|  Sequencer    |  |                                  | |  Controlled   |
|               |  |   MTC  (and MMC Responses)  *3   | |  Device       |
|               |  |<---------------------------------| |         *2    |
|_____|  |                                  | |_____|
```

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:
```
FO  7F  <ident>  <mcr>
    <SIGNATURE>  <count=2E>  01  00  00  00
    <count_1=14>
    7F  61  00  00  00  00  00  00  00  00
    7F  70  7F  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:
```
FO  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:
```
FO  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:
```
FO  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  .  .
            FO  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.

```
FO  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:

```
FO  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"):

```
FO  7F  <ident>  <mcr>  <RECORD STATUS>  <count=01>  00 F7
```

Finally, a Play command is issued, with MIDI Time Code turned on:

```
FO  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 . .
            FO  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
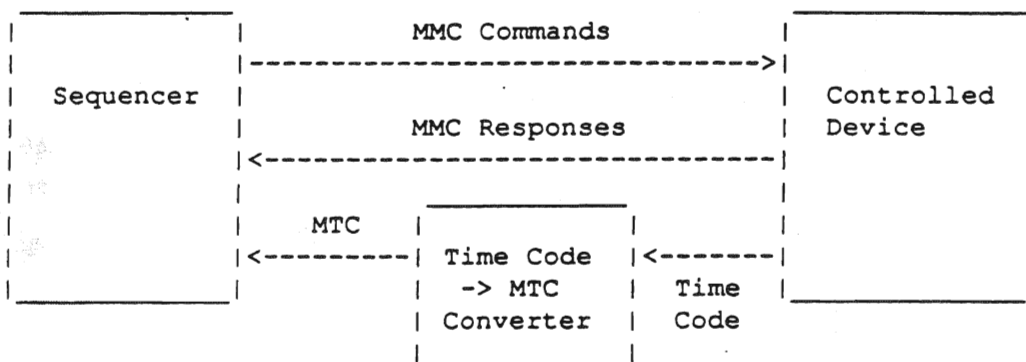
```
F1  0E . . F1 10 . . F1 29 . . F1 31 . .
F1  42 . .
            FO  7F  <ident>  <mcr>
                <RECORD STATUS>  <count=01> 00
            F7
            . . F1 50 . . F1 63 . . F1 74 . .
```

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.

```
 _____            MMC Commands            _____
|                |   _____                |                |
|                |  |------------------------------->|                |
|  Sequencer     |  |                                |  Controlled    |
|                |  |          MMC Responses         |  Device        |
|                |  |<-------------------------------|                |
|                |  |          _____      |                |
|                |  | MTC     |                |      |                |
|                |  |<--------| Time Code      |<-------|             |
|                |  |         |   -> MTC       | Time |                |
|_____|  |         | Converter      | Code |_____|
                               |_____|
```

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 *01*hex. The other device, the "Slave", has a device_ID of *02*hex. 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
3D 60 7F 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
```

83

```
             FO 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:
```
FO 7F <all-call=7F> <mcc> <WAIT> F7
```
.
.
.   (all transmissions halted)
.
.

Buffer clear:
```
FO 7F <all-call=7F> <mcc> <RESUME> F7
```

```
                 More master (00:22:05:16) and slave (10:01:59:02) time codes:
                 FO 7F <master=01> <mcr>
                     <Short SELECTED TIME CODE> 30 00
                 F7


                 FO 7F <slave=02> <mcr>
                     <Short SELECTED TIME CODE> 22 00
                 F7
```

Operator Stops the master:
```
FO 7F <master=01> <mcc> <STOP> F7
```

```
                 The master tally changes to reflect a "stopped" condition:
                 FO 7F <master=01> <mcr>
                     <MOTION CONTROL TALLY> <count=03> <STOP> 7F 01
                 F7
                 .
                 .
                 .
                 More slave time code (10:01:59:03):
                 FO 7F <slave=02> <mcr>
                     <Short SELECTED TIME CODE> 23 00
                 F7
                 .
                 .
                 .
                 Slave at 10:01:59:04:
                 FO 7F <slave=02> <mcr>
                     <Short SELECTED TIME CODE> 24 00
                 F7
```

Operator Stops the slave:
```
FO 7F <slave=02> <mcc> <STOP> F7
```

```
                 The slave tally shows "stopped":
                 FO 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:
```
FO  7F  <slave=02>  <mcc>
    <MOVE>  <count=02>
        <REQUESTED OFFSET>  <ACTUAL OFFSET>
F7
```

The Controller reads back the offset for display purposes:
```
FO  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):
```
FO  7F  <slave=02>  <mcr>
    <REQUESTED OFFSET>  69  27  35  12  00
F7
```

Operator pushes the "slave chase" button:
```
FO  7F  <slave=02>  <mcc>  <CHASE>  F7
```

The slave tally adjusts, showing chase mode, "stopped" and "parked" status:
```
FO  7F  <slave=02>  <mcr>
    <MOTION CONTROL TALLY>  <count=03>  <STOP>  <CHASE>  61
F7
```

Operator Plays the master (slave follows):
```
FO  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:
```
FO  7F  <master=01>  <mcr>
    <Short SELECTED TIME CODE>  38  00
    <MOTION CONTROL TALLY>  <count=03>  <PLAY>  7F  01
F7
```

```
FO  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:
```
FO  7F  <slave=02>  <mcc>
    <UPDATE>  <count=02>  <[BEGIN]>  <LOCK DEVIATION>
F7
```

The slave obliges (deviation = +5.02 frames):
```
FO  7F  <slave=02>  <mcr>
    <LOCK DEVIATION>  60  00  00  05  02
F7
    .
    .
    .
```

More master time (00:22:05:25):
```
FO  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:
```
FO  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):
```
FO  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:
```
FO  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:
```
FO  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.

86

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 the 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 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 most recent 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 im-plemented)	Set/reset as required, and if implemented.	Ignore d in WRITE data.
n (no time code)	1	$n = 0$ only after time code has been read from "tape".	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 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)	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]:

Time code Status Bits	Value after power up or MMC RESET	Value during Normal Operations
tt (time type)	See next column.	Follows tt in SELECTED TIME CODE, except that ACTUAL OFFSET and LOCK DEVIATION will remain non-drop-frame ($tt=11$) when SELECTED TIME CODE is drop-frame ($tt=10$).
c (color frame)	0	0
k (blank)	0	0
g (sign)	0	0/1 as required
i (final byte id)	0	0

06   GENERATOR 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 STAN-DARD or other internal default	tt = 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 tt. Subsequently determines the time code counting mode of the Generator.
c (color frame)	0	0	Ignore c in WRITE data.
k (blank)	0	0	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	0	Ignore e in WRITE data.
v (invalid code)	0	0	Ignore v in WRITE data.
d (video field 1)	0 (0/1 if im-plemented)	Set/reset as required, and if implemented.	Ignore d in WRITE data.
n (no time code)	0	0	Ignore n in WRITE data.

07   MIDI TIME CODE INPUT [read only]:

Time code Status Bits	Value after power up or MMC RESET	Value during Normal Operations
tt (time type)	TIME STAN-DARD or other internal default	If $n = 1$ (time code never read): tt = TIME STANDARD (or default) If $n = 0$: tt = As read from MTC
c (color frame)	0	0
k (blank)	1	$k = 0$ only after MTC has been received at the device's MIDI In. Otherwise, $k = 1$
g (sign)	0	0
i (final byte id)	1	1
e (estimated code)	0	0
v (invalid code)	0	Set as required.
d (video field 1)	0	0
n (no time code)	1	$n = 0$ only after MTC has been received at the device's MIDI In. Otherwise, $n = 1$ (i.e. $n = k$)

**08
thru
0F** GP0 thru GP7 [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 STAN- DARD or other internal default	tt = As loaded with WRITE or "Math" commands	OK to WRITE to tt.
c (color frame)	0	c = As loaded with WRITE or "Math" commands	OK to WRITE to c.
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	g = As loaded with WRITE or "Math" commands	OK to WRITE to g.

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

Appendix C   SIGNATURE TABLE

COMMAND BITMAP ARRAY:

Byte	Bit 6 (40h)	Bit 5 (20h)	Bit 4 (10h)	Bit 3 (08h)	Bit 2 (04h)	Bit 1 (02h)	Bit 0 (01h)
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:

Byte	Bit 6 (40h)	Bit 5 (20h)	Bit 4 (10h)	Bit 3 (08h)	Bit 2 (04h)	Bit 1 (02h)	Bit 0 (01h)
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

Byte	Bit 6 (40h)	Bit 5 (20h)	Bit 4 (10h)	Bit 3 (08h)	Bit 2 (04h)	Bit 1 (02h)	Bit 0 (01h)
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 :	MTC Cueing :
MMC "Events" trigger other MMC commands only.	MTC "Events" trigger event sequences, cues, and track punch in and out.
Events are defined by the EVENT command.	Events are defined by MTC Set-Up messages.
Each event is unique, and is identified by a single 7-bit name.	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.
Each Event definition specifies the source of the time code stream against which the Event should be triggered.	No time code source is specified, but is assumed to be MIDI Time Code.
Events contain additional flags for triggering at other than play speeds, and when moving forward or reverse or either.	No motion variations or restrictions are specified.
For compatibility with ESbus, each event may optionally be deleted after being triggered.	Events are not deleted when triggering occurs.
No overall event enable/disable is provided.	Event enable/disable commands turn all events on and off without affecting the Event List itself.
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.	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
```
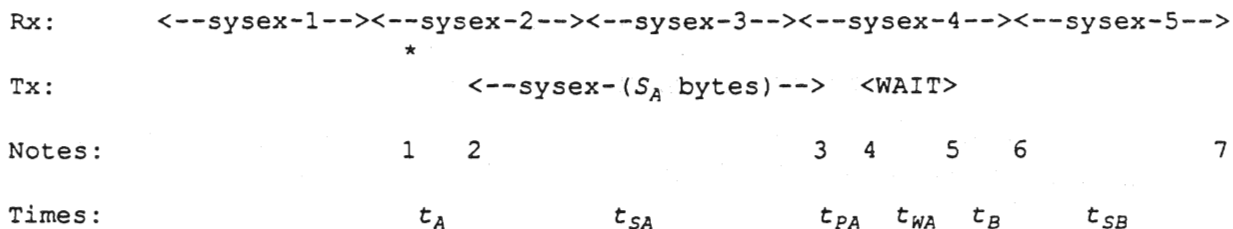
1.   A received byte causes device "A"'s receive buffer to fill beyond a predetermined threshold $\{H_A\}$, measured in bytes (see point "*" in the diagram).

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 x .320msec).

t_{Bmax} = 10msec. We thus establish a fairly generous requirement that <u>an MMC device must recognize the receipt of a WAIT message within 10msec</u>.

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:
$\quad H_A = 53$
$\quad t_{Amax} = \underline{10msec}$
$\quad t_{SAmax} = 53 \times (.320 + .100) = 22.26$ (i.e. <u>max delay between MIDI byte transmissions is 100usec.</u>)
$\quad t_{PAmax} = 0$
$\quad t_{WAmax} = 6 \times (.320 + .100) = 2.52$
$\quad X_A = 0$ (We temporarily ignore X_A 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

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

Second Printing 1998
Reformated 2014

MMA
POB 3173
La Habra CA 90632

CONTENTS

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 <u>body</u> 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 manufactures 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

Commands/Events that Affect RPN/NRPN Values (*survey*)	Respondents
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"):
$L(dB) = 40 \log (V/127)$ where V= CC#7 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:
$L(dB) = 40 \log (V/127^2)$ where V = (volume x 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 <u>after</u> the Pan command).

Findings:

Software makers seem to be more optimistic that sustained notes will be affected by Pan commands than the current state of 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 respondees 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 and 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 dis-similar 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* <u>*transmitters*</u> *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.

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 be 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 <u>all</u> 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:
    o Response to Reset All Controllers
    o Response to Data Increment/Decrement Controller
    o Sound Controller Defaults (Revised)
    o Renaming of CC91 and CC93
    o File Reference SysEx Message *(.pdf)*
    o Sample Dump Size/Rate/Name Extensions *(.pdf)*
    o Controller Destination SysEx Message *(.pdf)*
    o Key-based Instrument Controller SysEx Message *(.pdf)*
    o Global Parameter Control SysEx Message *(.pdf)*
    o Master Fine/Course Tuning SysEx Message *(.pdf)*
    o Redefinition of RPN01 and RPN02 (Channel Fine/Course Tuning)
    o RPN05 Modulation Depth Range *(.pdf)*
    o Extension 00-01 to File Ref SysEx Message *(.pdf)*
    o Default Pan Formula
    o High Resolution Velocity Prefix *(.pdf)*
    o Three Dimensional Sound Controllers *(.pdf)*

- These changes/additions were made to the SMF Specification since the "96.1 Second Edition" publication:
    o SMF Lyric Events Definition
    o SMF Device/Program Name Meta-events
    o SMF Language and Display Extensions
    o XMF Patch Prefix Meta Event

- These changes/additions were made to the MIDI Tuning Specification since the "96.1 Second Edition" publication:
    o Scale/Octave Tuning w/Defaults
    o 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:
    o Downloadable Sounds (DLS) (including DLS, DLS2, and Mobile DLS)
    o MIDI for IEEE-1394 *(pdf)*
    o eXtensible Music Format (XMF) (including XMF, mobile XMF, and related RPs)
    o Scalable Polyphony MIDI (SP-MIDI)
    o MIDI XML Names DTDs
    o Mobile Phone Control *(pdf)*
    o Mobile Musical Instrument *(pdf)*
    o 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.**