MIDI Machine Control 1.0

MMA0016 / RP013

MMA
POB 3173
La Habra CA 90632-3173

RP-013

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 <u>Information Fields</u> (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 <u>actual conditions</u> 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 <u>two</u> extension sets:

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

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

DATA LENGTHS

Since multiple Machine Control messages (i.e. Commands or Responses/Information Fields) may be transmitted within a single Sysex, and since the set of messages must be extendable in the future, it is necessary that any receiving device be able to determine the length of any received message, whether that message is known to the device or not.
Therefore, a large number of Commands and Information Fields include a byte count, while the remainder, in order to preserve bus bandwidth (particularly on the Response MIDI cable), have their length implied by the Hex value of their name byte.

COMMANDS:

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

RESPONSES/INFORMATION FIELDS:

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

NOTES:

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

SEGMENTATION

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

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

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

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

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

Exactly the same result could be achieved using segmentation:

```
F0 7F <device_ID> <mcc> <COMMAND SEGMENT> <count=05> 42 aa bb cc dd F7
F0 7F <device_ID> <mcc> <COMMAND SEGMENT> <count=05> 01 ee ff gg hh F7
F0 7F <device_ID> <mcc> <COMMAND SEGMENT> <count=04> 00 jj kk mm F7
```

ERROR HANDLING

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

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

COMMAND MESSAGE SYNTAX

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

<destination> ::= <device address>  |  <group address>  |  <all call address>
<device address> ::= 00  |  01  | ... |  7E
<group address> ::= 00  |  01  | ... |  7E
<all call address> ::= 7F

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

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

<command> ::= <command_code_0>
              | <command_code_variable> <count> <command data>
              | <handshake>

<command_code_0> ::= 01  |  02  | ... |  3F  |  00 <command_code_0>
<command_code_variable> ::= 40  |  41  | ... |  77  |  00 <command_code_variable>
<handshake> ::= 78  |  79  | ... |  7F  |  00 <handshake>
```

RESPONSE MESSAGE SYNTAX

```
<response message> ::= F0 7F <source> <mcr> <response string> F7

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

<mcr> ::= 07  [sub-ID 1 for MIDI Machine Control Responses]

<response string> ::= <response>  | <response string> <response>

<response> ::= <info_field_name_5> <standard 5-byte time code data>
               | <info_field_name_2> <2-byte short time code data>
               | <info_field_name_variable> <count> <info_field data>
               | <handshake>

<info_field_name_5> ::= 01  | 02  | ... | 1F  | 00 <info_field_name_5>
<info_field_name_2> ::= 20  | 21  | ... | 3F  | 00 <info_field_name_2>
<info_field_name_variable> ::= 40  | 41  | ... | 77  | 00 <info_field_name_variable>
<handshake> ::= 78  | 79  | ... | 7F  | 00 <handshake>
```

3 STANDARD SPECIFICATIONS

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

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

Two forms of Time Code subframe data are defined:
The first (labelled *{ff}*), contains subframe data exactly as described in the MIDI Cueing specification i.e. fractional frames measured in 1/100 frame units.
The second form (labelled *{st}*) substitutes time code "status" data in place of subframes. When processing time code data from a tape, for example, it is often useful to know whether "real" time code data is being received, or simply time data updated by the tape transport's tachometer pulses during a high speed wind.

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

```
hr mn sc fr {ff|st}

        hr = Hours and type: 0 tt 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)	
sss	Integer multiple of play-speed	.	Fractional part of play-speed	Integer range	Fractional 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,
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

<u>MOTION CONTROL PROCESS (MCP):</u>

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

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

<u>LOCAL TIME CODE (Time)</u>

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

<u>SYNCHRONIZATION (Sync)</u>

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

<u>GENERATOR (Gen)</u>

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

<u>MIDI TIME CODE (MTC)</u>

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 exit from record [rehearse].

01	STOP

02 PLAY (MCS command)

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

02	PLAY

NOTE:

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

03 DEFERRED PLAY (MCS command)

Identical to the PLAY command, with the exception that if the device is currently executing a LOCATE (MCP), then PLAY mode will not be invoked until the LOCATE is completed.
Receipt of any other MCS or MCP command will cancel DEFERRED PLAY.
When received while a LOCATE is in progress, the "MCP Success Level" field of the MOTION CONTROL TALLY Information Field must be set to indicate "Actively locating with Deferred Play pending" for the duration of the LOCATE.
When the LOCATE has concluded:
(i) An automatic PLAY (MCS) command will be issued;
(ii) The MOTION CONTROL TALLY "MCS command" byte will switch to "PLAY";
(iii) The MOTION CONTROL TALLY "MCP command" byte will become "No MCP's currently active", clearing both the LOCATE and Deferred Play indications.
If the device is not executing or does not support the LOCATE command, then it should immediately enter playback mode.

03	DEFERRED PLAY

NOTES:

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

04 FAST FORWARD (MCS command)

Move forward at maximum possible speed.
Output monitoring is controlled by the FAST MODE Information Field, if supported.
FAST FORWARD will be cancelled by the receipt of another MCS or MCP command.
Recording [rehearsing] tracks exit from record [rehearse].

04	FAST FORWARD

05 REWIND (MCS command)

Move in reverse direction at maximum possible speed.
Output monitoring is controlled by the FAST MODE Information Field, if supported.
REWIND will be cancelled by the receipt of another MCS or MCP command.
Recording [rehearsing] tracks exit from record [rehearse].

05	REWIND

06 RECORD STROBE

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

CONDITION 1: Controlled Device already Playing:

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

CONDITION 2: Controlled Device Stopped:

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

(i) An automatic PLAY (MCS) command will be issued; *3*4

(ii) At an appropriate point in the start up phase of the device, record [rehearse] entry will occur on all tracks which are presently in a record ready state. *1*5

06 RECORD STROBE

NOTES:

*1. Tracks are switched in and out of the record ready state using the TRACK RECORD READY Information Field.

2. It is recommended that, for new Controlled Device designs based on MMC, no recording [rehearsing] should take place following a RECORD STROBE command unless at least one track is in a record ready state.
Among existing non-MMC devices, however, it is quite common that, given that record [rehearse] has been enabled (for example by setting the appropriate value in the RECORD MODE Information Field), recording [rehearsing] will be initiated even if no tracks are in a record ready state when the command to record [rehearse] is received. Such operation remains permissible under MMC, provided that the resultant status is correctly indicated by including the "No Tracks Active" bit in the RECORD STATUS Information Field.

*3. Under CONDITION 2, an automatic PLAY (MCS) command will be issued only under the STOP or PAUSE conditions specified. At no other time does RECORD STROBE have any implications regarding play mode or playing speed.

*4. Also under CONDITION 2, the automatic PLAY (MCS) command will be issued whether or not any tracks are in a record ready state, and whether or not a RECORD MODE has been specified.

*5. The existence of a "record-pause" condition will not affect the operation of RECORD STROBE. (Refer to the RECORD PAUSE and PAUSE command descriptions.)

6. Devices which support and have their RECORD MODE set to "VTR:Insert" or "VTR:Assemble", are expected to produce clean and correctly timed transitions into record [rehearse] under both Conditions 1 and 2 outlined above. When starting from STOP or PAUSE mode (Condition 2), it will usually be necessary to wait until the device's start up phase has been completed before recording [rehearsing] is attempted.

7. A Controlled Device will ignore any RECORD STROBE command which is received while it is neither already in play mode nor completely stopped as described.

*8. Under CONDITION 1, "Controlled Device already Playing", it is not necessary for the PLAY or VARIABLE PLAY command to have been "successful" before RECORD STROBE is accepted. If, however, the desired play motion has not yet been achieved when RECORD STROBE is received, it may be necessary for the device to defer the onset of recording until an appropriate point in its start up phase.

*9. Note also that, under CONDITION 1, recording is not inhibited by Motion Control Process activity.

07 RECORD EXIT

Causes a record exit on all currently recording tracks.

07	RECORD EXIT

08 RECORD PAUSE

Causes the Controlled Device to enter "record-pause" mode provided that a PAUSE mode is already in effect (i.e. that the most recent Motion Control State is PAUSE). *4
No actual recording is initiated by RECORD PAUSE, but the device is placed in a state from which a transition into record mode can be made quickly and smoothly.
All "record ready" track Outputs are switched to monitor their respective Inputs.

A Controlled Device which supports the RECORD PAUSE command must fully implement the "record-pause" mode as described here and under the PAUSE command itself.
A Controlled Device which does not support this command must never enter "record-pause" mode.

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

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

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

08	RECORD PAUSE

NOTES:

1. RECORD PAUSE is the only way to enter "record-pause" from a non-record PAUSE without initiating motion.
2. RECORD PAUSE in itself will not produce a PAUSE.
3. The command sequence *<PAUSE> <RECORD PAUSE>* will always produce a paused state, and, if implemented, the "record-pause" condition.

*4. It is not necessary for the PAUSE command to have been "successful" in order for a RECORD PAUSE command to be accepted. Therefore, after receiving a *<PAUSE> <RECORD PAUSE>* command sequence, it may be necessary for the Controlled Device to defer the onset of "record-pause" until all motion has ceased following the PAUSE command. RECORD PAUSE must not cause any actual recording to take place.

5. RECORD PAUSE will be ignored by a Controlled Device if its Motion Control State is not already "PAUSE".

*6. Refer also to the RECORD STROBE and RECORD STROBE VARIABLE command definitions.

*7. This category includes PLAY, DEFERRED PLAY, VARIABLE PLAY and DEFERRED VARIABLE PLAY, none of which will cause recording to take place following a RECORD PAUSE.

8. No "rehearse-pause" mode is currently supported.

09 PAUSE (MCS command)

Stop as soon as possible.
VTR's and other visual devices stop "with picture".
The transport mechanism should be left in a state such that start up time will be minimized.

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

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

09	PAUSE

NOTES:

1. Repetition of the PAUSE command will <u>not</u> 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 <u>exit from record [rehearse]</u>.

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.

<u>MMC RESET must be supported by all MMC Controlled Devices.</u>

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 immediately:

(i) transmit the contents of the specified Information Field(s);

and (ii) add the specified Information Field name(s) to an internal UPDATE "list". *1

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

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

If any of the specified Information Fields are unsupported by the Controlled Device (or are undefined, or are defined as having "no access"), then a RESPONSE ERROR message will be generated naming the field(s) in error. (Valid names in the same UPDATE [BEGIN] request will be added to the UPDATE "list" in the usual way.) The RESPONSE ERROR message will be transmitted once, without re-transmissions. *4

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

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

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

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

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

NOTES:

*1. If a newly requested Information Field name is already contained in the internal UPDATE "list", then an immediate transmission of the contents of that field will occur as expected. The internal "list" will not change.

*2. If an Information Field value has changed more than once since the last UPDATE transmission, then only the most recent value will be transmitted.

*3. If an Information Field value has changed more than once since the last UPDATE transmission, but has reverted to the same value last transmitted, then no new UPDATE response will be required.

*4. Refer also to the RESPONSE ERROR Information Field description.

*5. Time code Information Field names may be specified either by their STANDARD TIME CODE names (01 thru 1F), or by their corresponding STANDARD SHORT TIME CODE names (21 thru 3F). Resulting UPDATE actions will be identical in either case.

FORMAT 2 - UPDATE [END]

The Controlled Device must delete the specified Information Field(s) from its UPDATE "list", and discontinue automatic re-transmissions of their contents.
No errors will be generated if the specified name(s) cannot be found in the current UPDATE list.

43	UPDATE [END]
<count=variable>	Byte count (not including command and count).
01	"END" sub-command
<name>	Information Field name(s).
.	*7F* anywhere in this list will discontinue <u>all</u> 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.)
<u>The result will always be expressed as a non-drop-frame quantity, regardless of the drop/non-drop status of either of the sources.</u>

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 Source #1, 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 drop-frame format (see "Drop Frame Notes" in the "Standard Specifications" section).
Take no action if the contents are not currently expressed in 30 frame, non-drop-frame format.
May be used after ADD or SUBTRACT to produce a drop frame result.
Valid Information Fields are the Read/Writeable fields in the 5-data-byte group (names 01 thru 1F).

4F	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.

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.

<u>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.</u>

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 <u>Variable</u> Play pending" for the duration of the LOCATE.

When the LOCATE has concluded:

(i) An automatic VARIABLE PLAY (MCS) command will be issued, and the device will enter continuously variable playback mode with the direction and speed specified (If the requested speed value exceeds the capabilities of the Controlled Device, then it should play at its "nearest available speed");

(ii) The MOTION CONTROL TALLY "MCS command" byte will switch to "VARIABLE PLAY";

(iii) The MOTION CONTROL TALLY "MCP command" byte will become "No MCP's currently active", clearing both the LOCATE and Deferred Variable Play indications.

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

`54`	DEFERRED VARIABLE PLAY
`<count=03>`	Byte count.
`sh sm sl`	Standard Speed Specification

NOTE:

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

55 RECORD STROBE VARIABLE

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

CONDITION 1: Controlled Device already Playing:

If the Controlled Device is already playing (i.e. the "Most recently activated Motion Control State" in the MOTION CONTROL TALLY Information Field is PLAY or VARIABLE PLAY), then RECORD STROBE VARIABLE will cause record [rehearse] entry on all tracks which are presently in a record ready state, and cause record [rehearse] exit on any currently recording [rehearsing] tracks that are no longer record ready. *2*9*10

CONDITION 2: Controlled Device Stopped:

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

(i) An automatic VARIABLE PLAY (MCS) command will be issued, and the device will enter continuously variable playback mode with the direction and speed specified. (If the requested speed value exceeds the capabilities of the Controlled Device, then it should play at its "nearest available speed"); *4*5

(ii) At an appropriate point in the varispeed start up phase of the device, record [rehearse] entry will occur on all tracks which are presently in a record ready state. *2*6

55	RECORD STROBE VARIABLE
<count=03>	Byte count.
sh sm sl	Standard Speed Specification

NOTES:

1. The recording [rehearsing] characteristics of RECORD STROBE VARIABLE are identical to those of the RECORD STROBE command. The only difference between the two commands lies in the nature of the play mode command which is invoked automatically if the Controlled Device is completely stopped. RECORD STROBE VARIABLE will therefore be used if variable speed recording [rehearsing] must be achieved from a standing start.

*2. Tracks are switched in and out of the record ready state using the TRACK RECORD READY Information Field.

3. It is recommended that, for new Controlled Device designs based on MMC, no recording [rehearsing] should take place following a RECORD STROBE VARIABLE command unless at least one track is in a record ready state.
Among existing non-MMC devices, however, it is quite common that, given that record [rehearse] has been enabled (for example by setting the appropriate value in the RECORD MODE Information Field), recording [rehearsing] will be initiated even if no tracks are in a record ready state when the command to record [rehearse] is received. Such operation remains permissible under MMC, provided that the resultant status is correctly indicated by including the "No Tracks Active" bit in the RECORD STATUS Information Field.

*4. Under CONDITION 2, an automatic VARIABLE PLAY (MCS) command will be issued only under the STOP or PAUSE conditions specified. At no other time does RECORD STROBE VARIABLE have any implications regarding play mode or playing speed.

*5. Also under CONDITION 2, the automatic VARIABLE PLAY (MCS) command will be issued whether or not any tracks are in a record ready state, and whether or not a RECORD MODE has been specified.

*6. The existence of a "record-pause" condition will not affect the operation of RECORD STROBE VARIABLE. (Refer to the RECORD PAUSE and PAUSE command descriptions.)

7. Devices which support and have their RECORD MODE set to "VTR:Insert" or "VTR:Assemble", are expected to produce clean and correctly timed transitions into record [rehearse] under both Conditions 1 and 2 outlined above. When starting from STOP or PAUSE mode (Condition 2), it will usually be necessary to wait until the device's start up phase has been completed before recording [rehearsing] is attempted.

8. A Controlled Device will ignore any RECORD STROBE VARIABLE command which is received while it is neither already in play mode nor completely stopped as described.

*9. Under CONDITION 1, "Controlled Device already Playing", it is not necessary for the PLAY or VARIABLE PLAY command to have been "successful" before RECORD STROBE VARIABLE is accepted. If, however, the desired play motion has not yet been achieved when RECORD STROBE VARIABLE is received, it may be necessary for the device to defer the onset of recording until an appropriate point in its start up phase.

*10. Note also that, under CONDITION 1, recording is not inhibited by Motion Control Process activity.

7C WAIT

Signals the Controlled Device that the Controller's receive buffer is filling (or that the Controller is otherwise unavailable), and that Machine Control Response transmissions must be discontinued until receipt of a RESUME from the Controller. Any Response transmission which is currently in progress will be allowed to proceed up to its normal End of System Exclusive (F7). Transmission of subsequent Responses may resume after receipt of a RESUME from the Controller.
The Responses WAIT and RESUME, however, are not inhibited by the WAIT Command. Neither is transmission of the WAIT Command itself inhibited by receipt of a WAIT Response.

A Controlled Device must guarantee:

(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. *F0 7F 7F <mcc> <WAIT> F7.*

7C	WAIT

NOTES:

1. Correct operation of the WAIT command requires a certain minimum size for the MIDI receive buffer in the Controller. Refer to Appendix E, "Determination of Receive Buffer Size".
2. Additional WAIT commands may be transmitted by a Controller should its receive buffer continue to fill.

7F RESUME

Signifies that the Controller is ready to receive Machine Control Responses from the Controlled Device. The default (power up) state is "ready to receive".
The RESUME command is used primarily to allow the Controlled Device to resume transmissions after a WAIT.
Transmission of the RESUME Command is not inhibited by the receipt of a WAIT Response.
The RESUME command is always the only command in its Sysex, and is directed to the "all-call" address i.e. *F0 7F 7F <mcc> <RESUME> F7.*

7F	RESUME

6 DETAILED RESPONSE & INFORMATION FIELD DESCRIPTIONS

Messages from CONTROLLED DEVICE to CONTROLLER.

00 Reserved for extensions

01 SELECTED TIME CODE [read/write]

Contains the time code normally used to reference the Controlled Device's current position. (It may also be referred to as "Self" code, or "Slave" time code.)
The Information Field SELECTED TIME CODE SOURCE determines the source of this time code. It is selected from Longitudinal Time Code (LTC), Vertical Interval Time Code (VITC), and the "tape counter" found on most tape machines.

01	SELECTED TIME CODE
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 <u>always expressed as a non-drop-frame number</u>.
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 <u>always expressed as a non-drop-frame number</u>.
ACTUAL OFFSET must be expressed in the range +/-12:00:00:00.00, based on the "time code equivalence" of numbers such as -01:00:00:00.00 and +23:00:00:00.00.

04	ACTUAL OFFSET
hr mn sc fr ff	Standard Time Specification with subframes (type *{ff}*)

NOTE:

Refer to Appendix B, "Time Code Status Implementation Tables" for exact usage of all embedded time code status bits as they apply to ACTUAL OFFSET.
Refer to Section 3, "Standard Specifications", for definition of these bits.

05 LOCK DEVIATION [read only]

For synchronization purposes, this field contains the time difference between the position of the Controlled Device's SELECTED TIME CODE and the SELECTED MASTER CODE after adjustment by the REQUESTED OFFSET:

LOCK DEVIATION = ACTUAL OFFSET - REQUESTED OFFSET
or
LOCK DEVIATION = SELECTED TIME CODE - SELECTED MASTER CODE - REQ'D OFFSET

LOCK DEVIATION is always a non-drop-frame number and must be expressed in the range +/-12:00:00:00.00.

05	LOCK DEVIATION
hr mn sc fr ff	Standard Time Specification with subframes (type *{ff}*)

NOTE:

Refer to Appendix B, "Time Code Status Implementation Tables" for exact usage of all embedded time code status bits as they apply to LOCK DEVIATION.
Refer to Section 3, "Standard Specifications", for definition of these bits.

06 GENERATOR TIME CODE [read/write]

Contains the current time code value being generated by the time code generator.

06	GENERATOR TIME CODE
hr mn sc fr st	Standard Time Specification with status (type *{st}*)

NOTE:

Refer to Appendix B, "Time Code Status Implementation Tables" for exact usage of all embedded time code status bits as they apply to GENERATOR TIME CODE.
Refer to Section 3, "Standard Specifications", for definition of these bits.

07 MIDI TIME CODE INPUT [read only]

Contains the most recent incoming MIDI Time Code value.

07	MIDI TIME CODE INPUT
hr mn sc fr st	Standard Time Specification with status (type *{st}*)

NOTE:

Refer to Appendix B, "Time Code Status Implementation Tables" for exact usage of all embedded time code status bits as they apply to MIDI TIME CODE INPUT.
Refer to Section 3, "Standard Specifications", for definition of these bits.

08 GP0 / LOCATE POINT [read/write]

General Purpose time code and calculation register 0.

08	GP0 / LOCATE POINT
hr mn sc fr ff	Standard Time Specification with subframes (type *{ff}*)

NOTES:

1. The LOCATE [I/F] command specifies that a General Purpose register must contain its target location time. Similarly, the EVENT command takes its trigger time from a General Purpose register. Therefore, at least one General Purpose register must be supported if either the LOCATE or the EVENT commands are to be used.
2. General Purpose registers may be employed to capture moving time code "on the fly" (for example by MOVE'ing the SELECTED TIME CODE to GPn). This can also remove the need for the Controller to always read back actual time code values, thus facilitating operation in the "open loop" mode.
3. Signed time code is permitted in a General Purpose register.
4. Refer to Appendix B, "Time Code Status Implementation Tables" for exact usage of all embedded time code status bits as they apply to the General Purpose registers.
   Refer to Section 3, "Standard Specifications", for definition of these bits.

09 GP1 [read/write]
0A GP2 [read/write]
0B GP3 [read/write]
0C GP4 [read/write]
0D GP5 [read/write]
0E GP6 [read/write]
0F GP7 [read/write]

General Purpose time code and calculation registers 1 thru 7.
(See notes for General Purpose register 0)

<name>	GP1 thru GP7
hr mn sc fr ff	Standard Time Specification with subframes (type *{ff}*)

21 Short SELECTED TIME CODE [read only]
22 Short SELECTED MASTER CODE [read only]
23 Short REQUESTED OFFSET [read only]
24 Short ACTUAL OFFSET [read only]
25 Short LOCK DEVIATION [read only]
26 Short GENERATOR TIME CODE [read only]
27 Short MIDI TIME CODE INPUT [read only]
28 Short GP0 / LOCATE POINT [read only]
29 Short GP1 [read only]
2A Short GP2 [read only]
2B Short GP3 [read only]
2C Short GP4 [read only]
2D Short GP5 [read only]
2F Short GP6 [read only]
2F Short GP7 [read only]

Refer to Section 3, "Standard Specifications", for definition of "Standard Short Time Code".
In each case, refer also to the corresponding 5-byte time code Information Field for description of data content.

<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 Command Bitmap Array
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 Response/Information Field Array
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".

*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

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 <u>not</u> 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 Time Code Jam 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 not be transmitted on the
                                     MMC Response cable, but may appear at
                                     other, unspecified, MIDI Out ports.
                         g = 0
<source>           Time Code Source:
                         00 = reserved for extensions
                         01 = SELECTED TIME CODE
                         02 = SELECTED MASTER CODE
                         06 = GENERATOR TIME CODE
                         07 = MIDI TIME CODE INPUT
                                (produces a "soft-THRU" mode)
                         7F = As locally defined [write only]
```

NOTES:

*1 In order to adequately track a high speed Time Code Source, and to guarantee correct MTC reception, these quarter frame messages should:

(a) run at the nominal frame rate;
and (b) be transmitted in "bursts", where each "burst" consists of several time code frames incrementing in a normal frame sequence.

2. The "time type" flags (`tt`) of the transmitted MIDI Time Code will be the same as those of the Time Code Source.

60 PROCEDURE RESPONSE [read only]

Allows the Controller to read back any, or all, of the assembled Procedures. The name of the Procedure to be read back must first be established by the PROCEDURE [SET] command.
If the PROCEDURE [SET] command specified "set all PROCEDURES", then a separate PROCEDURE RESPONSE will be transmitted for each Procedure currently assembled within the Controlled Device.
See also the PROCEDURE Command.

```
60                  PROCEDURE RESPONSE
<count=variable>    Byte count (not including command and count).
<procedure>         Procedure Name in the range 00 thru 7E.
                              7F = invalid Procedure set
                                        (i.e either the PROCEDURE [SET] command was
                                        never issued, or it specified an undefined Procedure,
                                        or there are currently no Procedures defined.)
                                        No further data required (<count=01>).
<command #1..>
<command #2..>
<command #3..>
       .
       .
       .
```

61 EVENT RESPONSE [read only]

Allows the Controller to read back any, or all, of the defined Events. The name of the Event to be read back must first be established by the EVENT [SET] command.
If the EVENT [SET] command specified "set all EVENT's", then a separate EVENT RESPONSE will be transmitted for each Event currently defined within the Controlled Device.
See also the EVENT Command.

```
61                  EVENT RESPONSE
<count=variable>    Byte count of following bytes.
<event>             Event Name in the range 00 thru 7E.
                              7F = invalid Event set
                                        (i.e either the EVENT [SET] command was never
                                        issued, or it specified an undefined Event, or there
                                        are currently no Events defined.)
                                        No further data required (<count=01>).
<flags>             Event control flags (see the EVENT [DEFINE] command for bit
                              definitions.)
<trigger source>    Information Field name of time code stream relative to which the event
                              is to be triggered (see the EVENT [DEFINE] command).
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.

TRACK MUTE overrides all other monitoring selections.

When read as a Response, the Controlled Device need transmit only as many bytes of TRACK MUTE as are required. Tracks not included in a Response transmission will be assumed not to be selected for individual muting. A Byte count of 00 may be used if no tracks are muted.
When written to by a WRITE command, tracks not included in the transmission will have their individual TRACK MUTE bits reset to zero (track unmuted). A Byte count of 00 may be used if all tracks are to be unmuted.

62	TRACK MUTE
<count=variable>	Byte count of following bitmap.
r0 r1 r2 . . .	Standard Track Bitmap (see Section 3). *2

NOTES:
1. The MASKED WRITE command may be used to change individual bits in the TRACK MUTE Information Field.
*2. TRACK MUTE is directed primarily at audio tracks. The "Video" bit will normally be zero.

63 VITC INSERT ENABLE [read/write]

Selects whether or not Vertical Interval Time Code is to be embedded in the video signal which is received at the Controlled Device's video input. If that video is subsequently to be recorded, then the VITC information will be recorded along with it.

VITC is derived from the Device's time code generator, and is therefore controlled also by the GENERATOR COMMAND and the GENERATOR SET UP Information Field. *1

63	VITC INSERT ENABLE
<count=03>	Byte count.
cc	Control: *00* = Disable *01* = Enable *7F* = As selected locally [Default/write only]
h1	First horizontal line number for VITC insertion; *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
```

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:

```
F0 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:

```
F0 7F <ident> <mcc>
    <WRITE> <count=03>
        <COMMAND ERROR LEVEL> <count=01> 3F
    <WRITE> <count=04>
        <MIDI TIME CODE SET UP> <count=02>
            00 <SELECTED TIME CODE>
F7
```

The operator requests Play. The Sequencer also requests that MIDI Time Code be turned on:

```
F0 7F <ident> <mcc>
    <RECORD EXIT> <PLAY>
    <MIDI TIME CODE COMMAND> <count=01> 02
F7
```

MIDI Time Code 03:02:20:28 (drop frame) arrives at the Sequencer's MIDI In:

```
F1 0C . . F1 11 . . F1 24 . . F1 31 . .
F1 42 . . F1 50 . . F1 63 . . F1 74 . .
```

The Sequencer checks that the device's internal copy of the current time code is in fact the same as the received MTC:

```
F0 7F <ident> <mcc>
    <READ> <count=01> <SELECTED TIME CODE>
F7
```

The device responds in the middle of the MTC frame 03:02:21:00-01:

```
F1 00 . . F1 10 . . F1 25 . . F1 31 . .
F1 42 . .
        F0 7F <ident> <mcr>
            <SELECTED TIME CODE> 43 02 15 21 00
        F7
      . . F1 50 . . F1 63 . . F1 74 . .
```

The operator "marks" on the fly a record punch in time of 03:02:27:08, which the Sequencer saves internally.

Some time later, the operator marks a record out point of 03:02:41:15, which the Sequencer also saves.

The operator then requests that the Sequencer rewind the transport and put it into record between the marked punch in and punch out points. Recording is to occur on tracks 1 and 2 only.
The Sequencer initiates a locate action, with MIDI Time Code turned off, and monitors MOTION CONTROL TALLY until the locate is complete:

```
F0 7F <ident> <mcc>
   <MIDI TIME CODE COMMAND> <count=01> 00
   <LOCATE> <count=06> <[TARGET]> 43 02 16 08 00
   <UPDATE> <count=02>
      <[BEGIN]> <MOTION CONTROL TALLY>
F7
```

The device responds immediately, showing that locating has begun in the reverse (rewind) direction:

```
F0 7F <ident> <mcr>
   <MOTION CONTROL TALLY> <count=03>
      <REWIND> <LOCATE> 01
F7
```

The device may pass through several different stages in the course of the locate:

```
F0 7F <ident> <mcr>
   <MOTION CONTROL TALLY> <count=03>
      <FAST FORWARD> <LOCATE> 00
F7
.
.
F0 7F <ident> <mcr>
   <MOTION CONTROL TALLY> <count=03>
      <FAST FORWARD> <LOCATE> 01
F7
.
.
F0 7F <ident> <mcr>
   <MOTION CONTROL TALLY> <count=03>
      <STOP> <LOCATE> 00
F7
.
.
F0 7F <ident> <mcr>
   <MOTION CONTROL TALLY> <count=03>
      <STOP> <LOCATE> 11
F7
```

After receiving the above "Locate complete" tally, the Sequencer ceases monitoring MOTION CONTROL TALLY, and then checks that the device has indeed located to the correct position:

```
F0 7F <ident> <mcc>
   <UPDATE> <count=02> <[END]> 7F
   <READ> <count=01> <SELECTED TIME CODE>
F7
```

The device's response, although representing a locate error of three frames, is deemed by the Sequencer to be satisfactory.

```
F0 7F <ident> <mcr>
   <SELECTED TIME CODE> 43 02 16 25 00
F7
```

The Sequencer now sets up all subsequent record activities, and chooses to monitor the RECORD STATUS information field in order to update its screen display:

```
F0 7F <ident> <mcc>
   <WRITE> <count=0F>
      <TRACK RECORD READY> <count=01> 60
      <GP1> 43 02 1B 08 00
      <GP2> 43 02 29 0F 00
   <EVENT> <count=06> <[DEFINE]> <event#=01>
      <flags=00> <SELECTED TIME CODE> <GP1>
      <RECORD STROBE>
   <EVENT> <count=06> <[DEFINE]> <event#=02>
      <flags=00> <SELECTED TIME CODE> <GP2>
      <RECORD EXIT>
   <UPDATE> <count=02> <[BEGIN]> <RECORD STATUS>
F7
```

The device immediately returns RECORD STATUS ("not recording"):

```
F0 7F <ident> <mcr> <RECORD STATUS> <count=01> 00 F7
```

Finally, a Play command is issued, with MIDI Time Code turned on:

```
F0 7F <ident> <mcc>
   <PLAY>
   <MIDI TIME CODE COMMAND> <count=01> 02
F7
```

At the record punch in point (03:02:27:08), the device inserts "recording" status between MTC messages: *1

```
F1 08 . .
       F0 7F <ident> <mcr>
          <RECORD STATUS> <count=01> 01
       F7
      . . F1 10 . . F1 2B . . F1 31 . .
F1 42 . . F1 50 . . F1 63 . . F1 74 . .
```

Similarly, "not recording" is returned at the punch out point (03:02:41:15): *1

```
F1 0E . . F1 10 . . F1 29 . . F1 31 . .
F1 42 . .
       F0 7F <ident> <mcr>
          <RECORD STATUS> <count=01> 00
       F7
      . . F1 50 . . F1 63 . . F1 74 . .
```

At the conclusion of recording, the transport is stopped, MIDI Time Code is turned off, as is monitoring of RECORD STATUS, and all tracks are returned to the "record not ready" state:

```
F0 7F <ident> <mcc>
    <STOP>
    <MIDI TIME CODE COMMAND> <count=01> 00
    <UPDATE> <count=02> <[END]> 7F
    <WRITE> <count=02>
        <TRACK RECORD READY> <count=00>
F7
```

NOTES:

*1 Although it is essentially the responsibility of the Controller (Sequencer) to keep the MIDI response line free of traffic during those periods when MTC timing is critical, an intelligent Controlled Device will make every attempt to prevent MMC Responses from causing jitter in the MTC messages.

*2. Whole systems of machines could be operated using this configuration. An intelligent translator would be required which would represent such a system as a single virtual machine.

*3. An additional MIDI In port at the Sequencer would make possible operations similar to this example while using an external Time Code to MTC converter. Control over MTC generation would be lost however, and the device would still need to be equipped with an internal time code reader.

```
 _____________                                   _____________
|             |         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 <u>all</u> command errors:

```
F0 7F <group=7C> <mcc>
   <WRITE> <count=06>
      <TIME STANDARD> <count=01> 03
      <COMMAND ERROR LEVEL> <count=01> <all=7F>
F7
```

The operator initiates a Play on the Master and then on the Slave.
(Note that this is not yet a CHASE operation):

```
F0 7F <master=01> <mcc> <PLAY> F7
F0 7F <slave=02> <mcc> <PLAY> F7
```

The Controller requests continuous updating of both time codes and motion tallies from both devices:

```
F0 7F <group=7C> <mcc>
   <UPDATE> <count=03> <[BEGIN]>
      <SELECTED TIME CODE>
      <MOTION CONTROL TALLY>
F7
```

Both master and slave respond with full 5 byte time code and tallies:
Master is PLAYing at 00:22:05:12 (00:16:05:0C hex).
Slave is PLAYing at 10:01:58:28 (0A:01:3A:1C hex).

```
F0 7F <master=01> <mcr>
   <SELECTED TIME CODE> 60 16 05 2C 00
   <MOTION CONTROL TALLY> <count=03> <PLAY> 7F 01
F7

F0 7F <slave=02> <mcr>
   <SELECTED TIME CODE> 6A 01 3A 3C 00
   <MOTION CONTROL TALLY> <count=03> <PLAY> 7F 01
F7
.
.
```

The next time code from the master (00:22:05:13) is in "short" form, as only the frames have changed. There has been no change at all in the MOTION CONTROL TALLY:

```
F0 7F <master=01> <mcr>
   <Short SELECTED TIME CODE> 2D 00
F7
```

Next slave code (10:01:58:29):

```
F0 7F <slave=02> <mcr>
   <Short SELECTED TIME CODE> 3D 00
F7
.
.
```

More master (00:22:05:14) and slave (10:01:59:00) times . . notice that the slave has seen a change in its "seconds", and has transmitted the entire 5 bytes of time code:

```
F0 7F <master=01> <mcr>
   <Short SELECTED TIME CODE> 2E 00
F7
```

```
F0 7F <slave=02> <mcr>
    <SELECTED TIME CODE> 6A 01 3B 20 00
F7
```

If the Controller's buffer were now filling up, it would transmit a WAIT request:

```
F0 7F <all-call=7F> <mcc> <WAIT> F7
```

.

.

. (all transmissions halted)

.

.

Buffer clear:

```
F0 7F <all-call=7F> <mcc> <RESUME> F7
```

More master (00:22:05:16) and slave (10:01:59:02) time codes:

```
F0 7F <master=01> <mcr>
    <Short SELECTED TIME CODE> 30 00
F7

F0 7F <slave=02> <mcr>
    <Short SELECTED TIME CODE> 22 00
F7
```

Operator Stops the master:

```
F0 7F <master=01> <mcc> <STOP> F7
```

The master tally changes to reflect a "stopped" condition:

```
F0 7F <master=01> <mcr>
    <MOTION CONTROL TALLY> <count=03> <STOP> 7F 01
F7
```

.

.

.

More slave time code (10:01:59:03):

```
F0 7F <slave=02> <mcr>
    <Short SELECTED TIME CODE> 23 00
F7
```

.

.

.

Slave at 10:01:59:04:

```
F0 7F <slave=02> <mcr>
    <Short SELECTED TIME CODE> 24 00
F7
```

Operator Stops the slave:

```
F0 7F <slave=02> <mcc> <STOP> F7
```

The slave tally shows "stopped":

```
F0 7F <slave=02> <mcr>
    <MOTION CONTROL TALLY> <count=03> <STOP> 7F 01
F7
```

Operator requests that the current difference between the master and slave positions become the slave's synchronization offset:

```
F0 7F <slave=02> <mcc>
    <MOVE> <count=02>
        <REQUESTED OFFSET> <ACTUAL OFFSET>
F7
```

The Controller reads back the offset for display purposes:

```
F0 7F <slave=02> <mcc>
    <READ> <count=01> <REQUESTED OFFSET>
F7
```

The slave responds with an offset of 09:39:53:18.00 (09:27:35:12.00 hex):

```
F0 7F <slave=02> <mcr>
    <REQUESTED OFFSET> 69 27 35 12 00
F7
```

Operator pushes the "slave chase" button:

```
F0 7F <slave=02> <mcc> <CHASE> F7
```

The slave tally adjusts, showing chase mode, "stopped" and "parked" status:

```
F0 7F <slave=02> <mcr>
    <MOTION CONTROL TALLY> <count=03> <STOP> <CHASE> 61
F7
```

Operator Plays the master (slave follows):

```
F0 7F <master=01> <mcc> <PLAY> F7
```

Master (00:22:05:24) and slave (10:01:59:09) return new tallies and time codes. The slave begins its synchronization process:

```
F0 7F <master=01> <mcr>
    <Short SELECTED TIME CODE> 38 00
    <MOTION CONTROL TALLY> <count=03> <PLAY> 7F 01
F7

F0 7F <slave=02> <mcr>
    <Short SELECTED TIME CODE> 29 00
    <MOTION CONTROL TALLY> <count=03> <PLAY> <CHASE> 01
F7
```

The operator chooses to observe the slave lock deviation (error) while synchronization is taking place:

```
F0 7F <slave=02> <mcc>
    <UPDATE> <count=02> <[BEGIN]> <LOCK DEVIATION>
F7
```

The slave obliges (deviation = +5.02 frames):

```
F0 7F <slave=02> <mcr>
    <LOCK DEVIATION> 60 00 00 05 02
F7
.
.
.
```

More master time (00:22:05:25):

```
F0 7F <master=01> <mcr>
    <Short SELECTED TIME CODE> 39 00
F7
```

Slave time (10:01:59:10) and deviation (-1.17 frames), not yet synchronized:

```
F0 7F <slave=02> <mcr>
    <Short SELECTED TIME CODE> 2A 00
    <Short LOCK DEVIATION> 41 17
F7
.
.
```

Slave time (10:02:01:00), deviation and tally (synchronization achieved):

```
F0 7F <slave=02> <mcr>
    <SELECTED TIME CODE> 6A 02 01 20 00
    <Short LOCK DEVIATION> 00 00
    <MOTION CONTROL TALLY> <count=03> <PLAY> <CHASE> 11
F7
```

With the Master in play, and the Slave synchronized, the operator now wishes to establish a small automatic "edit" by marking a Record "In" point and a Record "Out" point. The actual recording will be performed on the Slave machine.

From this point on, we shall only show significant UPDATE Responses.

Operator marks an "IN" point. The Controller captures the current time code by moving it into the GP1 general purpose register in both the master and slave machines using the group device_ID:

```
F0 7F <group=7C> <mcc>
    <MOVE> <count=02> <GP1> <SELECTED TIME CODE>
F7
.
.
```

Some time later, the operator marks an "OUT" point, and the Controller captures to the GP2 register:

```
F0 7F <group=7C> <mcc>
    <MOVE> <count=02> <GP2> <SELECTED TIME CODE>
F7
.
.
```

The operator now requests that the "edit" be performed automatically.

The Controller sets up events in the slave to punch into record at the time in GP1, and to punch out at time GP2. Both events are to be deleted after being triggered, and both are to be triggered only by forward motion play-speed time code.

The Controller also sets the slave's record mode and requests an automatic update of record status:

```
F0 7F <slave=02> <mcc>
   <EVENT> <count=06> <[DEFINE]> <event=#01>
      <flags=00> <SELECTED TIME CODE> <GP1>
      <RECORD STROBE>
   <EVENT> <count=06> <[DEFINE]> <event=#02>
      <flags=00> <SELECTED TIME CODE> <GP2>
      <RECORD EXIT>
   <WRITE> <count=03> <RECORD MODE> <count=01> 01
   <UPDATE> <count=02> <[BEGIN]> <RECORD STATUS>
F7
```

The slave immediately returns its current (non) record status:

```
F0 7F <slave=02> <mcr>
   <RECORD STATUS> <count=01> 00
F7
```

Next, the Controller sets up the master to (a) locate to a "preroll" point 5 seconds before the record punch in point in GP1 (GP0 is employed for the calculation); and (b) trigger an event to cause an automatic stop when the master reaches a point 2 seconds beyond the punch out point in GP2 (the trigger point is set up in GP3).

```
F0 7F <master=01> <mcc>
   <WRITE> <count=06> <GP0/LOCATE POINT> 60 00 05 00 00
   <SUBTRACT> <count=03> <GP0/LOCATE POINT> <GP1> <GP0/LOCATE POINT>
   <LOCATE> <count=02> <[I/F]> <GP0/LOCATE POINT>
   <WRITE> <count=06> <GP3> 60 00 02 00 00
   <ADD> <count=03> <GP3> <GP3> <GP2>
   <EVENT> <count=06> <[DEFINE]> <event=#01>
      <flags=10> <SELECTED TIME CODE> <GP3>
      <STOP>
F7
```

In addition to time codes and slave lock deviation, the following master and slave motion tallies will have been returned, indicating that the master is locating with the slave chasing:

```
F0 7F <master=01> <mcr>
   <MOTION CONTROL TALLY> <count=03>
      <REWIND> <LOCATE> 01
F7

F0 7F <slave=02> <mcr>
   <MOTION CONTROL TALLY> <count=03>
      <REWIND> <CHASE> 41
F7
.
.
.
```

Eventually the master will finish locating, but the slave may still be active:

```
F0 7F <master=01> <mcr>
    <MOTION CONTROL TALLY> <count=03>
        <STOP> <LOCATE> 11
F7

F0 7F <slave=02> <mcr>
    <MOTION CONTROL TALLY> <count=03>
        <REWIND> <CHASE> 41
F7
.
.
```

Finally, the slave parks with the master:

```
F0 7F <slave=02> <mcr>
    <MOTION CONTROL TALLY> <count=03>
        <STOP> <CHASE> 61
F7
```

Upon detecting that the machines have successfully located and parked, the Controller issues a play to the master, thus initiating the automated series of events previously loaded:

```
F0 7F <master=01> <mcc> <PLAY> F7
```

Master plays, slave attempts to synchronize:

```
F0 7F <master=01> <mcr>
    <MOTION CONTROL TALLY> <count=03> <PLAY> 7F 01
F7

F0 7F <slave=02> <mcr>
    <MOTION CONTROL TALLY> <count=03> <PLAY> <CHASE> 01
F7
.
.
```

Eventually, the slave is synchronized:

```
F0 7F <slave=02> <mcr>
    <MOTION CONTROL TALLY> <count=03> <PLAY> <CHASE> 11
F7
.
.
```

The slave punches into record at the pre-arranged point (and deletes Event #01):

```
F0 7F <slave=02> <mcr>
    <RECORD STATUS> <count=01> 01
F7
.
.
.
```

Some time later, the slave punches out (and deletes Event #02):

```
F0 7F <slave=02> <mcr>
    <RECORD STATUS> <count=01> 00
F7
.
.
```

Two seconds after the punch out, the master stops, causing the slave also to stop and park:

```
F0 7F <master=01> <mcr>
   <MOTION CONTROL TALLY> <count=03> <STOP> 7F 01
F7

F0 7F <slave=02> <mcr>
   <MOTION CONTROL TALLY> <count=03> <STOP> <CHASE> 61
F7
```

At the end of the session, the Controller mutes all update responses and disables both machines:

```
F0 7F <group=7C> <mcc>
   <UPDATE> <count=02> <[END]> <all=7F>
   <WRITE> <count=03> <CONTROL DISABLE> <count=01> 01
F7
```

Finally, the operator attempts to manipulate a (non-existent) time code generator in the master device:

```
F0 7F <master=01> <mcc>
   <READ> <count=01> <GENERATOR TIME CODE>
   <GENERATOR SET UP> <count=03>
      00 <SELECTED TIME CODE> 01
   <GENERATOR COMMAND> <count=01> 01
F7
```

The master first responds to the READ of the unsupported GENERATOR TIME CODE:

```
F0 7F <master=01> <mcr>
   <RESPONSE ERROR> <count=01> <GENERATOR TIME CODE>
F7
```

Then follows notification that the GENERATOR SET UP command is also unsupported. Since all errors have been enabled by the Controller, the master will enter "Error Halt" mode, and 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 STANDARD 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 implemented)	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 non-drop-frame (`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 STANDARD 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 implemented)	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 STANDARD 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 sl sm F7
                  (where sl 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:

$H_A = 53$
t_{Amax} = <u>10msec</u>
t_{SAmax} = 53 x (.320 + .100) = 22.26 (i.e. <u>max delay between MIDI byte transmissions is 100usec.</u>)
$t_{PAmax} = 0$
t_{WAmax} = 6 x (.320 + .100) = 2.52
$X_A = 0$ (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 wil! 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.