MIDI Show Control 1.1.1

Including 2-Phase Commit Enhancements

RP002/RP014

MMA
La Habra CA

MIDI Show Control (MSC) 1.1.1

1. Introduction

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

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

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

2. General Structure

2.1. Universal System Exclusive Format

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

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

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

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

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

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

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

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

2.2. Device Identification

<device_ID> is always a DESTINATION device address.

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

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

<device_ID> values:

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

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

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

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

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

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

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

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

2.3. Command_Formats

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

2.4. Commands

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

2.5. Extension Sets

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

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

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

2.6. Data Length

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

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

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

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

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

3. Standard Specifications

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

3.1. Cue Numbers

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

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

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

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

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

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

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

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

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

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

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

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

3.2. Time Code Numbers

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

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

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

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

Two forms of Time Code subframe data are defined:

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

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

```
hr mn sc fr (ff|st)
```

`hr` = Hours and type: `0 tt hhhhh`
- `tt` = time type (bit format):
  - 00 = 24 frame
  - 01 = 25 frame
  - 10 = 30 drop frame
  - 11 = 30 frame
- `hhhhh` = hours (0-23, encoded as 00H-17H)

`mn` = Minutes: `0 c mmmmmm`
- `c` = color frame bit (copied from bit in time code stream):
  - 0 = non color frame
  - 1 = color framed code
- `mmmmmm` = minutes (0-59, encoded as 00H-3BH)

`sc` = Seconds: `0 k ssssss`
- `k` = reserved - must be set to zero
- `ssssss` = seconds (0-59, encoded as 00H-3BH)

`fr` = Frames, byte 5 ident and sign: `0 g i fffff`
- `g` = sign bit:
  - 0 = positive
  - 1 = negative (where signed time code is permitted)
- `i` = final byte identification bit:
  - 0 = subframes
  - 1 = status
- `fffff` = frames (0-29, encoded as 00H-1DH)

If final byte bit = subframes (i = 0):
- `ff` = fractional frames: `0 bbbbbbb` (0-99, encoded as 00H-63H)

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

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

e = estimated code flag bit:

0 = normal time code

1 = tach or control track updated code

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

0 = valid

1 = invalid (error or not current)

d = video field identification bit:

0 = no field information in this frame

1 = first frame in 4 or 8 field video sequence

xxxx = reserved bits - must be set to 0000

Drop Frame Notes:

1. When writing time code data, the drop-frame or non-drop-frame status of the data being written may be overridden by the status of the Controlled Device (i.e. the time code from the device itself). For example, if the SET_CLOCK data are loaded with a non-drop-frame number and if the time code on the Controlled Device is drop-frame, then the SET_CLOCK data will simply be interpreted as a drop-frame number, with no attempt being made to perform any mathematical transformations.

2. Furthermore, if the above SET_CLOCK number had in fact been loaded with a non-existent drop-frame number (e.g. 00:22:00:00), then the next higher valid number would have been used (in this case, 00:22:00:02).

3. Calculation of offsets, or simply the mathematical difference between two time codes, can cause confusion when one or both of the numbers is drop-frame. For the purposes of this specification, DROP-FRAME NUMBERS SHOULD FIRST BE CONVERTED TO NON-DROP-FRAME BEFORE OFFSET CALCULATIONS ARE PERFORMED. Results of an offset calculation will then be expressed as non-drop-frame quantities.

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

4. Index List

4.1. Command_Formats

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

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

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

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

1. Less likelihood of errors in cueing. Digital communications can be demonstrated to be extremely reliable in repetitive duty conditions; much more so than tired or inexperienced stagehands.

2. More precise timing. Likewise, digital communications and computer control can be consistently accurate in automatic timing sequences and exactly as accurate as their human operators when under manual control.

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

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

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

Hex	command_format	Hex	command_format
00	reserved for extensions		
		40	Projection (General)
01	Lighting (General Category)	41	Film Projectors
02	Moving Lights	42	Slide Projectors
03	Color Changers	43	Video Projectors
04	Strobes	44	Dissolvers
05	Lasers	45	Shutter Controls
06	Chasers		
10	Sound (General Category)	50	Process Control (Gen.)
11	Music	51	Hydraulic Oil
12	CD Players	52	H_2O
13	EPROM Playback	53	CO_2
14	Audio Tape Machines	54	Compressed Air
15	Intercoms	55	Natural Gas
16	Amplifiers	56	Fog
17	Audio Effects Devices	57	Smoke
18	Equalizers	58	Cracked Haze
20	Machinery (General Cat.)	60	Pyro (General Category)
21	Rigging	61	Fireworks
22	Flys	62	Explosions
23	Lifts	63	Flame
24	Turntables	64	Smoke pots
25	Trusses		
26	Robots		
27	Animation		
28	Floats		
29	Breakaways		
2A	Barges		
30	Video (General Category)	7F	All-types
31	Video Tape Machines		
32	Video Cassette Machines		
33	Video Disc Players		
34	Video Switchers		
35	Video Effects		
36	Video Character Generators		
37	Video Still Stores		
38	Video Monitors		

4.2. Recommended Minimum Sets

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

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

The currently defined Recommended Minimum Sets are:

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

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

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

4.3. General Commands

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

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

4.4. Sound Commands

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

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

4.5. Two-Phase Commit Commands

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

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

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

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

5. Detailed Command And Data Descriptions

00 Reserved For Extensions

01 GO

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

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

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

02 STOP

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

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

03 RESUME

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

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

04 TIMED_GO

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

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

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

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

05 LOAD

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

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

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

06 SET

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

Standard Generic Control Numbers for Lighting

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

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

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

07 FIRE

07	FIRE
mm	Macro Number

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

08 ALL_OFF

08	ALL_OFF

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

09 RESTORE

09	RESTORE

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

0A RESET

0A	RESET

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

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

0B GO_OFF

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

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

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

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

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

10 GO/JAM_CLOCK

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

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

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

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

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

11 STANDBY_+

11	STANDBY_+
<Q_list>	optional

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

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

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

12 STANDBY_-

12	STANDBY_-
<Q_list>	optional

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

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

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

13 SEQUENCE_+

13	SEQUENCE_+
<Q_list>	optional

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

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

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

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

14 SEQUENCE_-

14	SEQUENCE_-
<Q_list>	optional

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

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

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

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

15 START_CLOCK

15	START_CLOCK
<Q_list>	optional

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

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

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

16 STOP_CLOCK

16	STOP_CLOCK
<Q_list>	optional

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

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

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

17 ZERO_CLOCK

17	ZERO_CLOCK
<Q_list>	optional

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

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

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

18 SET_CLOCK

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

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

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

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

19 MTC_CHASE_ON

19	MTC_CHASE_ON
<Q_list>	optional

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

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

1A MTC_CHASE_OFF

1A	MTC_CHASE_OFF
<Q_list>	optional

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

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

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

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

1B OPEN_CUE_LIST

1B	OPEN_CUE_LIST
<Q_list>	required

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

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

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

1C CLOSE_CUE_LIST

1C	CLOSE_CUE_LIST
<Q_list>	required

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

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

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

1D OPEN_CUE_PATH

1D	OPEN_CUE_PATH
<Q_path>	required

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

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

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

1E CLOSE_CUE_PATH

1E	CLOSE_CUE_PATH
<Q_path>	required

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

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

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

20 STANDBY

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

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

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

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

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

21 STANDING_BY

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

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

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

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

22 GO_2PC

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

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

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

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

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

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

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

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

23 COMPLETE

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

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

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

24 CANCEL

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

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

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

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

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

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

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

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

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

25 CANCELLED

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

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

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

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

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

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

26 ABORT

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

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

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

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

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

6. Two-Phase Commit Details

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

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

6.1. Controllers And Controlled Devices

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

6.2. Human Operators

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

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

6.3. Relating Two-Phase Commit To Other MSC Messages

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

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

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

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

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

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

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

6.4. Two-Phase Commit Message Sequences

6.4.1. Normal Message Sequences

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

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

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

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

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

6.4.2. Response Timeouts

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

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

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

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

6.4.3. Exceptional Condition Handling

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

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

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

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

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

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

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

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

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

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

1. Provision of the ABORT message to signal the presence of an exceptional condition and something about the nature of that condition (via the status code),

2. Usage of the CANCEL message to inform all participating controlled devices of a deviation from the error-free cue execution sequence, and

3. The large set of predefined status code values. These definitions allow the controller to display an informative description of the problem to the controller operator, who is most likely responsible for recovering from the problem.

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

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

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

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

6.4.4. Handling Exceptional Conditions With Message Retries

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

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

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

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

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

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

6.4.5. Manual Override Processing

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

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

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

6.4.6. Waiting For Messages

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

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

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

1. The amount of time that can elapse in such a wait is limited. The limit is 2 seconds for STANDING_BY and CANCELLED messages. The limit is variable (based on the contents of the previous STANDING_BY message) for COMPLETE messages.

2. A controller must be simultaneously waiting for numerous messages whose delivery order is indeterminate. Therefore, the controller must employ some form of table driven algorithm for tracking messages that are waiting for responses and detecting failures to receive responses within the specified limits.

6.5. Checksums

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

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

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

6.6. Sequence Numbers

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

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

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

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

6.7. Status Codes

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

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

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

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

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

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

The following table summarizes the status code ranges:

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

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

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

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

Status codes in CANCELLED messages

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

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

Status codes in ABORT messages

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

6.9. Examples Of Two-Phase Commit Usage

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

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

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

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

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

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

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

MIDI Two-Phase Commit Example Cue

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

The sequence in which these cues occur is:

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

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

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

6.9.1. Basic Message Exchanges

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

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

Error-Free MIDI Two-Phase Commit Example Cue Execution

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

Notes:

A. Send STANDBY messages for all cues that will go before the turntable completes 90 degrees of rotation. N.B. this must be done early enough to allow the full two second timeout interval to elapse before any actual cue execution is necessary.

B. STANDING_BY response messages received. Only sequence numbers are used to identify the cues to which the STANDING_BY messages apply. Also, the STANDING_BY messages may not be received in the same order that the STANDBY messages were sent. In this case, the flys

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

C. The electric eye STANDBY aborts because the actress has not broken the electric eye beam.

D. The electric eye STANDING_BY message indicates that the actress has broken the electric eye beam. (Strict MIDI two-phase commit completeness requires that a CANCEL be sent for EE-6. But this is unnecessary since the controlled device does not require it.)

E. Note the seconds column. There is an approximately three second delay between receipt of the STANDING_BY message at 06.010 and initiation of the flys and lights cues at 09.000.

F. The flys cue 28.1 GO_2PC message is sent before the COMPLETE message is received for cue 28. The flys controller interprets this as an indication that line 12 should be kept moving while the COMPLETE message for cue 28 is sent.

G. Once flys cue 28 reports complete, the turntable cue 34 and light cue 118.1 can be started. At this point, the standby message exchanges for the remaining cues also are performed. The same timeout considerations discussed in note A above apply here.

H. Line 12 is completely flown at this point.

I. The turntable cue 34.1 GO_2PC message is sent before the COMPLETE message is received for cue 34. Like the flys controlled device, the turntable controlled device interprets this as an indication that motion should be continued while the COMPLETE message for cue 34 is sent.

J. Once turntable cue 34 reports complete, sound cue 110 can be started.

K. Once turntable cue 34.1 reports complete, light cue 119 can be started.

6.9.2. Error Condition Detected Early

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

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

Notes:

A. The first flys cue returns the ABORT message with a "motor failure" status. This status would be converted to text and displayed to the controller operator.

B. CANCEL messages are sent to all controlled devices for all cues that have not yet completed execution. This conforms to the two-phase commit error recovery principles described in Section 6.4. Because the individual controlled devices are still processing the STANDBY messages, STANDING_BY and CANCEL messages are intermixed on the wire.

C. The CANCELLED messages begin arriving. Like STANDING_BY messages, the CANCELLED messages need not be received in the same order that the CANCEL messages were sent. The CANCELLED status is "terminated" because none of the canceled cues ever began execution.

D. The controller left nothing to chance. It sent a CANCEL message for F-28.1 even though an ABORT message for that cue was received at 00.013 seconds. Since the flys controlled device had already "forgotten" that F-28.1 was standing by, its CANCELLED message contains a "not standing by" status code.

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

6.9.3. Errors Detected During Execution Of A Cue

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

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

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