

TIMAX2 COMMAND SET SPECIFICATION

Revision 1.0.8

Prepared By: Timothy G. Bartoo

TiMaxSpatial
Focusrite Plc.

Contents

1 Version History	5
2 Introduction	5
3 Ethernet Protocols	5
3.1 Ethernet Subnets	5
3.2 IP Address Acquisition	5
4 Ethernet Commands	6
4.1 Channel Numbers	6
4.2 Channel Number Range	6
4.3 Multi-byte Integers	7
4.4 UNIT COMMANDS	8
4.4.1 REBOOT	8
4.4.2 SET DEFAULTS	8
4.4.3 SET MIDI CHANNEL	9
4.4.4 SET IP ADDRESS	9
4.4.5 SET TIME	9
4.4.6 SET UNIT NAME	9
4.4.7 TRANSMIT MIDI	9
4.4.8 TRANSMIT UDP	10
4.5 MATRIX COMMANDS	11
4.5.1 CLEAR CROSSPOINT ROW DELAY	12
4.5.2 CLEAR CROSSPOINT ROW GAIN	12
4.5.3 MUTE ALL OUTPUTS	12
4.5.4 RECALL IMAGE DEFINITION	12
4.5.5 RECALL IMAGE DEFINITION WITH TRACKING	12
4.5.6 SET ALL OUTPUT DELAY	12
4.5.7 SET ALL IO LEVELS	13
4.5.8 SET ALL IO CARD SOURCE	13
4.5.9 SET ALL NETWORK SOURCE	13
4.5.10 SET ALL PLAYBACK SOURCE	14
4.5.11 SET CROSSPOINT DELAY	14
4.5.12 SET CROSSPOINT LEVEL	14
4.5.13 SET CROSSPOINT ROW DELAY	14
4.5.14 SET CROSSPOINT ROW LEVEL	15
4.5.15 SET IMAGE DEFINITION	15
4.5.16 SET INPUT LEVEL	15
4.5.17 SET INPUT MUTE	15
4.5.18 SET INPUT SOLO	16
4.5.19 SET MATRIX	16
4.5.20 SET OUTPUT DELAY	16
4.5.21 SET OUTPUT LEVEL	16
4.5.22 SET OUTPUT MUTE	17
4.5.23 SET OUTPUT SOLO	17
4.5.24 SET PANSOURCE POINT	18
4.5.25 SET PANSOURCE POINT WITH TRACKING	18

4.5.26 SET SOLO MODE	18
4.5.27 SET SOURCE MIX	19
4.5.28 SET SIGNAL INVERT	19
4.5.29 SET TRACKING ENABLES	19
4.5.30 SET TRACKING RECALL RAMP	19
4.5.31 SET TRIGGER ENABLES	21
4.5.32 ZERO DELAY	21
4.6 GROUP COMMANDS	22
4.6.1 ASSIGN GROUP	22
4.6.2 CLEAR ALL GROUPS	22
4.6.3 CLEAR GROUP	22
4.6.4 CLEAR GROUPS	22
4.6.5 SET GROUP LEVEL	22
4.6.6 SET GROUP MUTE	22
4.6.7 SET GROUP SOLO	23
4.6.8 ZERO GROUPS	23
4.7 EQ COMMANDS	24
4.7.1 CLEAR EQ CHANNELS	25
4.7.2 CLEAR EQ BANDS	25
4.7.3 SET EQ	25
4.8 SHOW CONTROLLER COMMANDS	26
4.8.1 GO CUE	26
4.8.2 OPEN SHOW	26
4.8.3 SET CUE CLOCK	26
4.8.4 STOP ALL	26
4.8.5 STOP ALL CUES	26
4.8.6 STOP CUE	26
4.8.7 STOP RAMPS	27
4.8.8 RUN MIDI TIMECODE GENERATOR	27
4.8.9 SET MIDI TIMECODE GENERATOR	27
4.8.10 START MIDI TIMECODE GENERATOR	27
4.8.11 STOP MIDI TIMECODE GENERATOR	27
4.8.12 RUN SHOW CLOCK	27
4.8.13 SET SHOW CLOCK	28
4.8.14 START SHOW CLOCK	28
4.8.15 STOP SHOW CLOCK	28
4.9 PLAYBACK COMMANDS	29
4.9.1 PLAYBACK LOAD	29
4.9.2 PLAYBACK LOAD NUMBER	29
4.9.3 PLAYBACK LOAD LOOP	29
4.9.4 PLAYBACK GO	30
4.9.5 PLAYBACK GO LOOP	30
4.9.6 PLAYBACK GO SET	30
4.9.7 PLAYBACK PAUSE	30
4.9.8 PLAYBACK QUEUE	30
4.9.9 PLAYBACK RESUME	31
4.9.10 PLAYBACK STOP LOOPING	31
4.9.11 PLAYBACK UNLOAD	31

4.10 GET STATUS COMMANDS	32
4.10.1 AUTOUPDATE	32
4.10.2 GET CROSSPOINT DELAYS	32
4.10.3 GET CROSSPOINT LEVELS	32
4.10.4 GET DIRECTORY	33
4.10.5 GET DISK STATE	35
4.10.6 GET IO LEVELS	35
4.10.7 GET EQ	36
4.10.8 GET GROUP ASSIGNS	36
4.10.9 GET METERING	38
4.10.10 GET MODULE	38
4.10.11 GET MUTE SOLO	39
4.10.12 GET OUTPUT DELAYS	39
4.10.13 GET PLAYBACK	40
4.10.14 GET REVERB	40
4.10.15 GET REVERB EQ PARAM	40
4.10.16 GET REVERB LEVELS	41
4.10.17 GET REVERB MOD TABLE	41
4.10.18 GET REVERB OUTPUT GAIN	42
4.10.19 GET STATUS	42
4.10.20 GET TRACKING ENABLES	43
4.10.21 INQUIRY	44
4.11 REVERB COMMANDS	45
4.11.1 SET REVERB DELAY LINE INPUT GAIN	45
4.11.2 SET ALL REVERB DELAY LINE INPUT GAIN	45
4.11.3 SET REVERB DELAY MODULATOR CONTROL	45
4.11.4 SET REVERB DELAY MODULATOR ENTRIES	45
4.11.5 SET REVERB DELAY MODULATOR ENTRY	46
4.11.6 SET REVERB ENABLE	46
4.11.7 SET REVERB FDN DELAY	46
4.11.8 SET ALL REVERB FDN DELAY	46
4.11.9 SET REVERB FDN FILTER COEFFICIENT	46
4.11.10 SET ALL REVERB FDN FILTER COEFFICIENT	47
4.11.11 SET REVERB INPUT GAIN	47
4.11.12 SET REVERB MATRIX GAIN	47
4.11.13 SET REVERB MIX FADER GAIN	47
4.11.14 SET REVERB MATRIX FEEDBACK GAIN	47
4.11.15 SET ALL REVERB MATRIX FEEDBACK GAIN	48
4.11.16 SET REVERB OUTPUT GAIN	48
4.11.17 SET REVERB OUTPUT GAINS	48
4.11.18 SET REVERB PRE FEEDBACK GAIN	48
4.11.19 SET REVERB POST DELAY	48
4.11.20 SET REVERB PRE DELAY	49
4.11.21 SET REVERB PREFILTER COEFFICIENTS	49
4.11.22 SET REVERB PRE POST FLAGS	49
4.11.23 SET REVERB PARAMETERS	49
4.11.24 REVERB RECALL	50
4.12 NOTIFICATIONS	51

5 MIDI Commands	52
5.1 MIDI Messages	52
5.2 MIDI Channel	52
5.3 MIDI Command Table	53
5.4 Programming a MIDI Trigger	54
6 File Transfer	55
6.1 File Transfer Routines	55
6.1.1 Initialization: initializeFileTransfer()	55
6.1.2 Sending Files: transmitFile()	55
6.1.3 Receiving Files: receiveFile()	55
6.2 File Types	56

1 Version History

Issue	Description	By	Date
1.0.0	First Draft.	TGB	2024-07-05
1.0.1	Added SET REVERB DELAY MODULATOR ENTRIES.	TGB	2024-07-11
1.0.2	Corrected 4.10.1, 4.10.3, 4.10.12, 4.10.14	TGB	2024-07-22
1.0.3	Added SET REVERB OUTPUT GAINS command	TGB	2024-07-22
1.0.4	Added File Transfer commands	TGB	2024-10-29
1.0.5	Added four reverb commands, which set all delay lines on an engine	TGB	2025-02-21
1.0.7	Added new reverb commands	TGB	2025-11-20
1.0.8	Upgraded documentation for GO CUE command	TGB	2026-03-03

2 Introduction

This document specifies the TiMax command set for reference by software developers.

There are a small number of commands in the set which are not specified in this document which are specific to TiMax2 software internal functions: key management, etc.

3 Ethernet Protocols

The TiMax unit receives ethernet UDP packets sent to its current IP address and to port 0xE7C3 (59,331).

Software can discover TiMax units on its local area network by broadcasting an [INQUIRY](#) command to the TiMax port (0xE7C3).

3.1 Ethernet Subnets

A TiMax unit and an ethernet device, such as a computer, must be on the same subnet for communication to take place. A “255” in the subnet mask means that that part of the address must match. For example, if the subnet mask is “255.255.255.0”, addresses 192.168.1.33 and 192.168.1.123 are on the same subnet.

3.2 IP Address Acquisition

Unless set to use a fixed IP, a TiMax unit during boot up uses the DHCP protocol to request a dynamically-assigned IP address from a DHCP server on the local area network. If no response is received from a DHCP server, the unit will choose an unused IP address in the self-assign range (169.254.x.x) after a short delay. These methods enable the unit to acquire a unique local area IP address without user intervention.

When IP addresses on a local area network are manually-managed, a fixed IP address can be assigned to a TiMax unit via

1. the TiMax2 unit’s front panel
2. the Information / Configuration window Hardware tab in TiMax2 software, or
3. by sending a [SET IP ADDRESS](#) command.

When a fixed IP has been set, the unit does not use DHCP and does not self-assign an address during boot up.

4 Ethernet Commands

This section specifies formatting details for commands sent via ethernet. Command formats specified in this section comprise the data portion of a UDP packet. One TiMax2 command is sent per UDP packet. The command string length is 288 bytes or less.

TiMax2 commands are strings of bytes, short integers (2 bytes: 16-bits) and long integers (4 bytes: 32-bits). In this document, integers within TiMax2 commands are colour-coded as follows:

byte: 1A, short integer: 1A2B, long integer: 1A2B3C4D.

In this document, constants are always shown in hexadecimal.

A TiMax2 command always begins with a long integer command code.

Command parameters (if any) occupy the data following the command code. For parameters, symbolic names are used rather than hexadecimal constants. For example, a short integer amplitude parameter could appear as:

ampl, and a long integer group number parameter could appear as: group.

4.1 Channel Numbers

In the command set, channel numbers are zero-based. Software typically labels the channel number range, for example, as 1 through 64 for display to users, so there is an “off-by-one” situation between the numbers in the command set and the corresponding numbers displayed to the user.

Some command codes include a channel number in the least-significant byte. For example, the command code for `SET PANSPACE POINT` is shown as: 100200cc. If this command is being sent to input channel 18, the hexadecimal constant 0x09020000 would have 18 added to it before being written into a TiMax2 command. The resulting command code would be 0x09020012.

4.2 Channel Number Range

For TiMax2 commands, the input and output channel range is typically 0 to 63. However, for a given TiMax unit, the implemented channel range depends on hardware configuration. The `INQUIRY` message, which is sent when a TiMax unit connects to software, includes the implemented channel range for that unit. Software must ensure that commands are generated that conform to the implemented channel range for the unit that is connected.

4.3 Multi-byte Integers

In commands and in returned data structures, short integers and long integers are sent in big-endian byte order for compatibility with the Coldfire processor executing the TiMax2 firmware. A call to `htons(x)` (“host to network short”) converts a short integer `x` (16-bits) on any computer platform to a big-endian short integer for inclusion in a TiMax command. A call to `htonl(y)` (“host to network long”) converts a long integer `y` (32-bits) on any platform to a big-endian long integer for inclusion in a TiMax command. These functions `htons()` and `htonl()` are typically present in any C-language compilation environment.

Short integers and long integers received in any TiMax2 data structure are also in big-endian byte order. A call to `ntohs(x)` (“network to host short”) converts a big-endian short integer `x` in a TiMax data structure to a short integer on any computer platform. A call to `ntohl(x)` (“network to host long”) converts a big-endian long integer `x` in a TiMax data structure to a long integer on any computer platform. These functions `ntohs()` and `ntohl()` are typically present in any C-language compilation environment.

Single byte integers require no conversion.

4.4 UNIT COMMANDS

4.4.1 REBOOT

8D060000	A3F42C19
----------	----------

REBOOT forces the TiMax2 unit to immediately reboot. Power amplifiers to which the TiMax2 are connected should be muted prior to executing this command.

All multi-byte integers are [big-endian](#).

4.4.2 SET DEFAULTS

5C040000

SET DEFAULTS sets the TiMax2 unit to factory default settings:

- sets all input and output gains off
- sets all crosspoint gains off
- sets all source mix levels to 0 dB
- clears all mutes and solos
- sets all delay settings to 0
- clears EQs on all channels
- clears all group assignments
- sets all group faders to zero dB
- sets no show loaded during boot up
- sets DHCP for getting IP address
- clears unit password (no password required)
- clears all digital audio network settings
- clears unit name
- clears unit key
- enables MIDI input on all MIDI channels
- sets GPI lockout to 1 second
- sets GPI trigger mode to playback with address debounce

All multi-byte integers are [big-endian](#).

4.4.3 SET MIDI CHANNEL

4D040000	ch	set	0	0
----------	----	-----	---	---

The SET MIDI CHANNEL command sets the MIDI channel (0 to 15) for MIDI communications. Incoming MIDI messages containing a matching MIDI channel in the four least-significant-bits of the status byte are received by the unit; other messages are ignored.

If the channel `ch` and `set` parameters are both zero, the command sets the unit to receive on any MIDI channel. If the channel `ch` parameter is set to a specific MIDI channel 0 to 15 inclusive and the `set` parameter is 1, the unit is set to receive only on MIDI channel `ch`.

All multi-byte integers are [big-endian](#).

4.4.4 SET IP ADDRESS

0D060000	adrs
----------	------

SET IP ADDRESS sets the fixed IP for the unit to use for ethernet communications. When a fixed IP is set, the unit receives packets addressed to this address, and identifies packets it sends with this address. If the fixed IP is set to zero (no fixed IP is set), the unit negotiates with a DHCP server on the local area network for the IP address to use. If no DHCP server responds, after a time, the unit self-assigns an IP address.

The `adrs` parameter is set to the desired fixed IP address, or if `adrs` is zero, the fixed IP address is cleared.

All multi-byte integers are [big-endian](#).

4.4.5 SET TIME

83060000	00	yr	mn	d	dw	h	m	s
----------	----	----	----	---	----	---	---	---

SET TIME sets the real time clock. The `yr` parameter contains year minus 2000. The `mn` parameter contains month, 0 through 11 inclusive. The `d` parameter contains day, 0 through 31 inclusive. The `dw` parameter contains day of week, 1 through 7 inclusive, with 1 equal to Monday. The `h` parameter contains hour, 0 through 23 inclusive. The `m` parameter contains minute, 0 through 59 inclusive. The `s` parameter contains second, 0 through 59 inclusive.

All multi-byte integers are [big-endian](#).

4.4.6 SET UNIT NAME

86060000	n0	...	nn	00000000
----------	----	-----	----	----------

SET UNIT NAME sets the name string for the unit. This name is displayed in the unit selection popup menu. If no name is set, the unit serial number is used as the name.

The unit name is included as a UTF-8 string. The first character of the name string is shown as `n0` and the last character of the string is shown as `nn`. The string is terminated by a zero long integer. The maximum string length (before termination long integer) is 16 bytes.

All multi-byte integers are [big-endian](#).

4.4.7 TRANSMIT MIDI

1C040000	port	len	b0	...	bn	00000000
----------	------	-----	----	-----	----	----------

TRANSMIT MIDI transmits a counted string on one of the two MIDI OUT ports. The MIDI port to transmit on is in the `port` parameter: 0 for port 1 and 1 for port 2. The length of the string is in the `len` parameter. The first byte of the MIDI string is shown as `b0` and the last character of the string is shown as `bn`. The string is terminated by a zero long integer. The maximum string length (before termination long integer) is 264 bytes.

All multi-byte integers are [big-endian](#).

4.4.8 TRANSMIT UDP

0B040000	IP	port	len	b0	...	bn	00000000
----------	----	------	-----	----	-----	----	----------

TRANSMIT UDP transmits a counted string on as a UDP packet addressed to the specified IP address `IP` and port `port`. The length of the string is in the `len` parameter. The first byte of the UDP string is shown as `b0` and the last character of the string is shown as `bn`. The string is terminated by a zero long integer. The maximum string length (before termination long integer) is 260 bytes.

All multi-byte integers are [big-endian](#).

4.5 MATRIX COMMANDS

The TiMax2 64-channel delay matrix mixer () contains 4,096 crosspoints. Matrix gain and delay settings are implemented at the crosspoints. Output delay settings affect the delay value in all crosspoints on the output channel's column.

The input signals are fed into 64 / 128 delay lines. For each crosspoint, two delay line taps, designated A and B, select signals from the delay line. At each crosspoint, the gain and delay of the signal is controlled.

A Level Control provides the control signal for crosspoint gain. Crosspoint delay is changed using some combination of single-sample and region crossfades. For both types of crossfade, a ramp generator provides complimentary linear control signals to the multipliers on each tap. When no delay crossfade is underway, the control signal for the currently-active tap is '1' and the control signal for the currently-inactive tap is '0'.

The currently-inactive tap can be freely moved without affecting the crosspoint output signal. When a crossfade starts, the currently-inactive tap is moved to the new target delay value, and the ramp generator is then run in the direction that crossfades from the previous delay value to the new delay value.

A single-sample crossfade is used to add or remove delay at a rate of up to 3.9 milliseconds of delay per second. This translates to maximum of one sample added or subtracted from the crosspoint delay every 256 samples. Beyond this rate, audible pitch shifts occur. The CPU initiates a single-sample crossfade by setting an enable bit. The FPGA begins the single-sample crossfade immediately (it is not necessary to analyze the incoming signal). The currently-inactive tap is set to one sample before (delay going down) or after (delay going up) the currently-active tap and a 256-sample crossfade is started. When the crossfade completes the enable bit is cleared. The CPU knows the single-sample crossfade will complete 256 sample cycles (plus a few setup cycles) after setting the enable bit (the enable bit can be write-only).

The more-complex method is a region crossfade. This method repeats or removes an audio region, typically hundreds of samples in length, by crossfading between two delay line taps separated by the region length. The duration of the delay crossfade is fixed at 2048 samples.

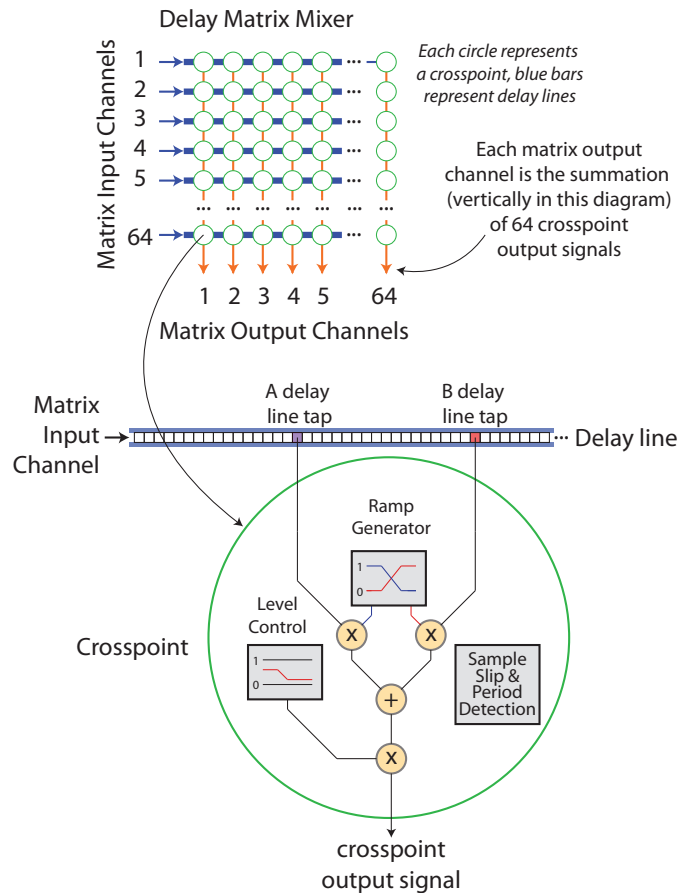


Figure 1: Crosspoints

4.5.1 CLEAR CROSSPOINT ROW DELAY

520400cc

The CLEAR CROSSPOINT ROW DELAY command sets to zero all crosspoint delay values for an input channel. The input channel is set in the lower byte *cc* of the [command code](#).

All multi-byte integers are [big-endian](#).

4.5.2 CLEAR CROSSPOINT ROW GAIN

510400cc

The CLEAR CROSSPOINT ROW GAIN command sets to zero all crosspoint gain (amplitude) values for an input channel. The input channel is set in the lower byte *cc* of the [command code](#).

All multi-byte integers are [big-endian](#).

4.5.3 MUTE ALL OUTPUTS

43040000	state
----------	-------

The MUTE ALL OUTPUTS command turns on and off the mute state of all output channels.

State is one to mute all outputs and zero to unmute all outputs.

All multi-byte integers are [big-endian](#).

4.5.4 RECALL IMAGE DEFINITION

260200cc	index	ramp
----------	-------	------

The RECALL IMAGE DEFINITION command recalls an image definition that is defined in an open show onto an input channel. The input channel *cc* is set in the lower byte of the [command code](#). The image definition is identified by its tracking number, as shown in the XML show file (e.g. `trackn="2"`). This command recalls the image definition as long as it exists in the open show, regardless of any tracking enable settings.

Ramps are expressed in units of milliseconds. The crosspoint amplitude and delay values on the channel ramp from their current values to the values specified in the image definition over the specified time.

If *ramp* is set to -1 (FFFFFFFF), the ramp time and ramp enables set in [SET TRACKING RECALL RAMP](#) are used.

All multi-byte integers are [big-endian](#).

4.5.5 RECALL IMAGE DEFINITION WITH TRACKING

260A00cc	index	ramp
----------	-------	------

The RECALL IMAGE DEFINITION WITH TRACKING command is identical to RECALL IMAGE DEFINITION (immediately above), except that this version of the command requires tracking to be enabled for the input channel if the image definition will be recalled. See [SET TRACKING ENABLES](#).

All multi-byte integers are [big-endian](#).

4.5.6 SET ALL OUTPUT DELAY

18040000	00000001	d0	...	dn	0F00
----------	----------	----	-----	----	------

The SET ALL OUTPUT DELAY command sets all output delay values. The command contains a variable-length list of delay values *d0* through *dn*. The length of the list is 64 or less. This list is applied to channel 0

through n . Delay values for output channels $n + 1$ and above are zeroed. An empty list zeros the delay on all output channels.

Delay values d_0 through d_n are expressed in units of 0.1 mS. The maximum delay value is 9999, which is 999.9 mS, just under one second.

All multi-byte integers are [big-endian](#).

4.5.7 SET ALL IO LEVELS

17040000	ramp	0	a0	...	an	0F00
----------	------	---	----	-----	----	------

Sets all input levels

17040000	ramp	1	a0	...	an	0F00
----------	------	---	----	-----	----	------

Sets all output levels

The SET ALL IO LEVELS command sets all input or output amplitude values. Ramps are expressed in units of milliseconds.

The command contains a variable-length list of amplitude values a_0 through a_n . The length of the list is 64 or less. This list is applied to channel 0 through n . Amplitudes for channels $n + 1$ and above are zeroed. An empty list zeros the gain on all channels.

Amplitude values are expressed in units of 0.1 dB, with an offset of 1000. An amplitude value of 1000 is 0.0 dB; 900 is -10.0 dB. The maximum amplitude value at a group is 1100, which is +10.0 dB. An amplitude value of zero is off ($-\infty$ dB).

All multi-byte integers are [big-endian](#).

4.5.8 SET ALL IO CARD SOURCE

12040000	a0	...	an	0F00
----------	----	-----	----	------

The SET ALL IO CARD SOURCE command sets all IO card source mix values. IO Card inputs come physically from the IO cards plugged into the right-side of the chassis (viewed from the rear).

The command contains a variable-length list of amplitude values a_0 through a_n . The length of the list is 64 or less. This list is applied to channels 0 through n . Amplitudes for channels $n + 1$ and above are zeroed. An empty list zeros the IO card source mix level on all channels.

Amplitude values are expressed in units of 0.1 dB, with an offset of 1000. An amplitude value of 1000 is 0.0 dB; 900 is -10.0 dB. The maximum amplitude value at a group is 1100, which is +10.0 dB. An amplitude value of zero is off ($-\infty$ dB).

All multi-byte integers are [big-endian](#).

4.5.9 SET ALL NETWORK SOURCE

14040000	a0	...	an	0F00
----------	----	-----	----	------

The SET ALL NETWORK SOURCE command sets all audio network source mix values. Network audio channels are transported on digital audio networks, for example, Dante.

The command contains a variable-length list of amplitude values a_0 through a_n . The length of the list is 64 or less. This list is applied to channels 0 through n . Amplitudes for channels $n + 1$ and above are zeroed. An empty list zeros the audio network source mix level on all channels.

Amplitude values are expressed in units of 0.1 dB, with an offset of 1000. An amplitude value of 1000 is 0.0 dB; 900 is -10.0 dB. The maximum amplitude value at a group is 1100, which is +10.0 dB. An amplitude value of zero is off ($-\infty$ dB).

All multi-byte integers are [big-endian](#).

4.5.10 SET ALL PLAYBACK SOURCE

13040000	a0	...	an	0F00
----------	----	-----	----	------

The SET ALL PLAYBACK SOURCE command sets all playback source mix values. Playback audio channels are streamed from the unit's disk into the matrix.

The command contains a variable-length list of amplitude values a_0 through a_n . The length of the list is 64 or less. This list is applied to channels 0 through n . Amplitudes for channels $n + 1$ and above are zeroed. An empty list zeros the playback source mix level on all channels.

Amplitude values are expressed in units of 0.1 dB, with an offset of 1000. An amplitude value of 1000 is 0.0 dB; 900 is -10.0 dB. The maximum amplitude value at a group is 1100, which is +10.0 dB. An amplitude value of zero is off ($-\infty$ dB).

All multi-byte integers are [big-endian](#).

4.5.11 SET CROSSPOINT DELAY

16040000	input	output	delay	ramp
----------	-------	--------	-------	------

The SET CROSSPOINT DELAY command sets the delay for a crosspoint.

Delay values are expressed in units of 0.1 mS. The maximum delay value is 9999, which is 999.9 mS, just under one second.

Ramps are expressed in units of milliseconds. The delay ramps from its current value to the new, specified value over the specified time. Commands generated from user-interface slider controls typically specify a 10 millisecond ramp.

All multi-byte integers are [big-endian](#).

4.5.12 SET CROSSPOINT LEVEL

3A040000	input	output	ampl	ramp
----------	-------	--------	------	------

The SET CROSSPOINT LEVEL command sets the amplitude level of a crosspoint.

Amplitude values are expressed in units of 0.1 dB, with an offset of 1000. An amplitude value of 1000 is 0.0 dB; 900 is -10.0 dB. The maximum amplitude value at a group is 1100, which is +10.0 dB. An amplitude value of zero is off ($-\infty$ dB).

Ramps are expressed in units of milliseconds. Commands generated from user-interface slider controls typically specify a 10 millisecond ramp.

All multi-byte integers are [big-endian](#).

4.5.13 SET CROSSPOINT ROW DELAY

1E040000	ramp	start	end	d0	...	dn	F000
----------	------	-------	-----	----	-----	----	------

The SET CROSSPOINT ROW DELAY command sets the delays for a range of crosspoint rows. The command contains a variable-length list of delay values. This array is used to set all the crosspoint delays on input channels $start$ through end inclusive. If the channel range $start$ through end inclusive is larger than the number of delay values in the list, upper channels are zeroed.

Delay values are expressed in units of 0.1 mS. The maximum delay value is 9999, which is 999.9 mS, just under one second.

All multi-byte integers are [big-endian](#).

4.5.14 SET CROSSPOINT ROW LEVEL

1D040000	ramp	start	end	a0	...	an	F000
----------	------	-------	-----	----	-----	----	------

The SET CROSSPOINT ROW LEVEL command sets the amplitudes for a range of crosspoint rows. The command contains a variable-length list of amplitude values. This array is used to set all the crosspoint amplitudes on input channels *start* through *end* inclusive. If the channel range *start* through *end* inclusive is larger than the number of amplitude values in the list, upper channels are zeroed.

Amplitude values are expressed in units of 0.1 dB, with an offset of 1000. An amplitude value of 1000 is 0.0 dB; 900 is -10.0 dB. The maximum amplitude value at a group is 1100, which is +10.0 dB. An amplitude value of zero is off ($-\infty$ dB). All multi-byte integers are [big-endian](#).

4.5.15 SET IMAGE DEFINITION

090200cc	ramp	a0	...	a63	d0	...	d63	r0	...	r3	TiMax2-only
----------	------	----	-----	-----	----	-----	-----	----	-----	----	-------------

The SET IMAGE DEFINITION command sets the amplitudes and delays for every output associated with the specified input channel. In other words, this command sets a matrix row. The input channel *cc*, 0 through 63 inclusive, is set in the lower byte of the [command code](#).

This command is similar to [SET PANSPLACE POINT](#), except that the input channel's location on the Panspace is unaffected.

An image definition completely specifies the matrix settings for the specified input channel. An image definition consists of an array of 64 matrix amplitude values, an array of 64 matrix delay values, and an array of four reverb engine values. All of the values in the arrays in an image definition are short integers.

The amplitudes and delays ramp from their current level to the levels specified in this command over the specified ramp time. Ramps are expressed in units of milliseconds.

Amplitude values (*a0*..*a63*) are expressed in units of 0.1 dB, with an offset of 1000. An amplitude value of 1000 is 0.0 dB; 900 is -10.0 dB. The maximum amplitude value at a crosspoint is 1000, which is 0.0 dB. An amplitude value of zero is off ($-\infty$ dB).

Delay values (*d0*..*d63*) are expressed in units of 0.1 mS. The maximum delay value is 9999, which is 999.9 mS, just under one second.

All multi-byte integers are [big-endian](#).

4.5.16 SET INPUT LEVEL

410300cc	ramp	ampl
----------	------	------

The SET INPUT LEVEL command sets the amplitude level of an input channel. The amplitude ramps from its current level to the new, specified level over the specified time. The input channel is set in the lower byte *cc* of the [command code](#).

Ramps are expressed in units of milliseconds. The amplitude ramps from its current level to the new, specified level over the specified ramp time. Commands generated from user-interface slider controls typically specify a 10 millisecond ramp.

Amplitude values are expressed in units of 0.1 dB, with an offset of 1000. An amplitude value of 1000 is 0.0 dB; 900 is -10.0 dB. The maximum amplitude value at an input is 1100, which is +10.0 dB. An amplitude value of zero is off ($-\infty$ dB).

All multi-byte integers are [big-endian](#).

4.5.17 SET INPUT MUTE

450400cc	state
----------	-------

The SET INPUT MUTE command turns on and off the mute state of an input channel. The input channel is set in the lower byte *cc* of the [command code](#).

State is one to turn mute on and zero to turn mute off.

All multi-byte integers are [big-endian](#).

4.5.18 SET INPUT SOLO

470400cc	state
----------	-------

The SET INPUT SOLO command turns on and off the solo state of an input channel. The input channel is set in the lower byte *cc* of the [command code](#).

State is one to turn solo on and zero to turn solo off.

All multi-byte integers are [big-endian](#).

4.5.19 SET MATRIX

4C040000	code
----------	------

The SET MATRIX command zeros all delays in the matrix, and sets all the gains in the matrix according to the *code*.

If *code* is 0, all input, output and crosspoint gains are set to zero. This is the “zero the matrix” command.

If *code* is 1, all input and output gains are set to 0 dB, all diagonal crosspoint gains are set to 0 dB and all non-diagonal crosspoint gains are set to zero. A diagonal crosspoint has identical input and output channel numbers, i.e. crosspoint {0,0}, {1,1}, {2,2}, etc. This sets up the matrix so all input channels are connected at 0 dB to the output channel of the same number. This is often useful for testing purposes.

All multi-byte integers are [big-endian](#).

4.5.20 SET OUTPUT DELAY

1B0400cc	dly
----------	-----

The SET OUTPUT DELAY command sets the delay on an output channel. The output channel is set in the lower byte *cc* of the [command code](#).

Delay value *dly* is expressed in units of 0.1 mS. The maximum delay value is 9999, which is 999.9 mS, just under one second.

All multi-byte integers are [big-endian](#).

4.5.21 SET OUTPUT LEVEL

420400cc	ramp	ampl
----------	------	------

The SET OUTPUT LEVEL command sets the amplitude level of an output channel. The output channel is set in the lower byte *cc* of the [command code](#).

Ramps are expressed in units of milliseconds. The amplitude ramps from its current level to the new, specified level over the specified ramp time. Commands generated from user-interface slider controls typically specify a 10 millisecond ramp.

Amplitude values are expressed in units of 0.1 dB, with an offset of 1000. An amplitude value of 1000 is 0.0 dB; 900 is -10.0 dB. The maximum amplitude value at an output is 1100, which is +10.0 dB. An amplitude value of zero is off ($-\infty$ dB).

All multi-byte integers are [big-endian](#).

4.5.22 SET OUTPUT MUTE

460400cc	state
----------	-------

The SET OUTPUT MUTE command turns on and off the mute state of an output channel. The output channel is set in the lower byte *cc* of the [command code](#).

State is one to turn mute on and zero to turn mute off.

All multi-byte integers are [big-endian](#).

4.5.23 SET OUTPUT SOLO

480400cc	state
----------	-------

The SET OUTPUT SOLO command turns on and off the solo state of an output channel. The output channel is set in the lower byte *cc* of the [command code](#).

State is one to turn solo on and zero to turn solo off.

All multi-byte integers are [big-endian](#).

4.5.24 SET PANSPACE POINT

100200cc	ramp	xCoord	yCoord	zCoord	subsp			
a0	...	a63	d0	...	d63	r0	...	r3

The SET PANSPACE POINT command sets the amplitudes and delays for every output associated with the specified input channel. In other words, this command sets a matrix row. It also sets the Panspace location of an input channel, which is displayed on a Panspace display.

Panspace point command parameters are: channel, ramp, x coordinate, y coordinate, z coordinate, subspace number and (for TiMax2) an image definition.

The command contains the image definition that is to be applied, and the spatial coordinates are only used in generating location updated messages.

The input channel is set in the lower byte *cc* of the [command code](#).

Panspace point coordinates are expressed in decimetre (0.1 metre) units, from 0 to 2896 (maximum is 289.6 metres).

An image definition completely specifies the matrix settings for the specified input channel. An image definition consists of an array of 64 matrix amplitude values, an array of 64 matrix delay values, and an array of four reverb engine values. All of the values in the arrays in an image definition are short integers.

The amplitudes and delays ramp from their current level to the levels specified in this command over the specified ramp time. Ramps are expressed in units of milliseconds.

In TiMax2, amplitude values (*a0* . . . *a63*) are expressed in units of 0.1 dB, with an offset of 1000. An amplitude value of 1000 is 0.0 dB; 900 is -10.0 dB. The maximum amplitude value at a crosspoint is 1000, which is 0.0 dB. An amplitude value of zero is off ($-\infty$ dB). Delay values (*d0* . . . *d63*) are expressed in units of 0.1 mS. The maximum delay value is 9999, which is 999.9 mS, just under one second.

In TiMax2 software, a function called `imageDefForPoint()` generates the image definition for this command, based on the three coordinates, the subspace and the image definitions that have been placed on the image definition layer of the Panspace. The source code for this function is available. The image definitions placed on the Panspace can be automatically calculated by the TiMax2 software and saved to the show file. The locations of the image definitions placed on the panspace can be read out from the XML file for the show that is currently open.

The subspace number *subsp* specifies the subspace that the panspace point should reference. A subspace in the TiMax software contains a user-defined subset of the complete set of image definitions that have been placed on the image definition layer of the Panspace. If no subspaces have been defined, then there is only one subspace: the default “All” subspace which always contains the complete set of image definitions. In this case, the subspace number should be one. If subspaces have been defined, then the show file XML should be opened and the subspace list located (search on <subspaces>). The list is in the order of subspace number, starting at one.

All multi-byte integers are [big-endian](#).

4.5.25 SET PANSPACE POINT WITH TRACKING

100A00cc	ramp	xCoord	yCoord	zCoord	subsp	a0	...	a63	d0	...	d63	r0	...	r3
----------	------	--------	--------	--------	-------	----	-----	-----	----	-----	-----	----	-----	----

The SET PANSPACE POINT WITH TRACKING command is identical to SET PANSPACE POINT (immediately above), except that this version of the command requires tracking to be enabled for the input channel if the image definition will be recalled. See [SET TRACKING ENABLES](#).

All multi-byte integers are [big-endian](#).

4.5.26 SET SOLO MODE

74060000	mode
----------	------

The SET SOLO MODE command sets the solo mode of the matrix. In Single-channel Solo Mode, soloing a channel clears all solos except for the channel being soloed. In Multi-channel Solo Mode (“additive solo mode”), soloing a channel adds it to the list of soloed channels.

The command parameter mode is one to set Single-channel Solo Mode and zero to set Multi-channel Solo Mode. All multi-byte integers are [big-endian](#).

4.5.27 SET SOURCE MIX

1A0400cc	ioc	plbk	mod
----------	-----	------	-----

The SET SOURCE MIX command sets the three source mix amplitude settings for an input channel. The input channel is set in the lower byte *cc* of the [command code](#).

Each input channel receives three signal sources, IO card, Playback and Network module. IO card signals are from cards plugged into the right-hand side of the TiMax2 frame: analogue or AES3. Playback signals are audio files being played back from the TiMax disk drive. Network module signals are incoming from an audio network, such as Dante, MADI, etc.

The *ioc* parameter sets the IO card mix amplitude value. The *plbk* parameter sets the playback mix amplitude value. The *mod* parameter sets the audio network mix amplitude value.

Amplitude values are expressed in units of 0.1 dB, with an offset of 1000. An amplitude value of 1000 is 0.0 dB; 900 is -10.0 dB. The maximum amplitude value at a crosspoint is 1000, which is 0.0 dB. An amplitude value of zero is off ($-\infty$ dB).

All multi-byte integers are [big-endian](#).

4.5.28 SET SIGNAL INVERT

3B0400cc	inv
----------	-----

The SET SIGNAL INVERT command inverts the signal on an input or output channel. Inverting a digital audio signal is done by multiplying the signal by -1. The channel is set in the lower byte *cc* of the [command code](#). In TiMax2, input channels are 0 to 63 and output channels are 64 to 127.

If *inv* is 1, the signal is inverted, if *inv* is 0, the signal is not inverted.

All multi-byte integers are [big-endian](#).

4.5.29 SET TRACKING ENABLES

0C040000	c0	...	cn	00000000
----------	----	-----	----	----------

The SET TRACKING ENABLES command sets the per-channel tracking enable flags. This command contains a variable-length list, *c0* ... *cn* of input channel numbers. The list can be in any order. The list is terminated with a long integer zero. Tracking is enabled for all input channels in the list and disabled for all input channels not in the list.

All multi-byte integers are [big-endian](#).

4.5.30 SET TRACKING RECALL RAMP

49040000	gre	dre	rtime
----------	-----	-----	-------

The SET TRACKING RECALL RAMP command sets a ramp time and ramp enables which are sometimes used when an image definition is recalled via its tracking number.

There are two cases in which the settings made by this command are used:

1. When the [RECALL IMAGE DEFINITION](#) command is executed with -1 passed as that command’s ramp parameter.

2. When a MIDI continuous controller message is received on an enabled MIDI channel on MIDI port 2. This will recall the image definition whose tracking number is in byte 3 of the MIDI message onto the input channel in byte 2 of the MIDI message, using the ramps enabled by this command.

In either case, the image definition recalled is present in the show currently open in the TiMax2 unit. The image definition is identified by its tracking number, as shown in the XML show file (e.g. `trackn="2"`). When a show is saved, the current contents of the image definition library is written into the show.

The 'gain ramp enabled' `gre` parameter enables (1) or disables (0) gain ramps. The 'delay ramp enabled' `dre` parameter enables (1) or disables (0) delay ramps. The ramp time `rtime` parameter sets the ramp time in milliseconds.

All multi-byte integers are [big-endian](#).

4.5.31 SET TRIGGER ENABLES

32040000	trk	plbk	cue
----------	-----	------	-----

The SET TRIGGER ENABLES command sets three global trigger enables:

1. Tracking enable `trk`: This enables (1) or disables (0) the recall of image definitions via MIDI continuous controller message received on MIDI port 2.
2. Playback enable `plbk`: This enables (1) or disables (0) the playback engine. When disabled, no playback will occur.
3. Cue enable `cue`: This enables (1) or disables (0) the triggering of cues in the show controller. When disabled, no cues will be triggered.

All multi-byte integers are [big-endian](#).

4.5.32 ZERO DELAY

4B040000

The ZERO DELAY command zeros all output and crosspoint delays.

All multi-byte integers are [big-endian](#).

4.6 GROUP COMMANDS

4.6.1 ASSIGN GROUP

1F040000	group	c1	...	cn	FF	00000000
----------	-------	----	-----	----	----	----------

The ASSIGN GROUP command assigns input and output channels to a group, overwriting the previous channel assignment for the group.

The group number is 0 to 31.

The list of channels to be assigned to the group are represented above by *c1* ... *cn*. Channel numbers for this command are in the range 0 to 127 inclusive. Input channels are 0 to 63 and output channels are 64 to 127. The list is terminated by a constant FF, and then terminated by a long zero.

All multi-byte integers are [big-endian](#).

4.6.2 CLEAR ALL GROUPS

54040000

The CLEAR ALL GROUPS command clears all channels from all groups.

All multi-byte integers are [big-endian](#).

4.6.3 CLEAR GROUP

56040000	group
----------	-------

The CLEAR GROUP command clears all input and output channels from a group.

The group number is 0 to 31.

All multi-byte integers are [big-endian](#).

4.6.4 CLEAR GROUPS

53040000	group
----------	-------

The CLEAR GROUPS command clears all input and output channels from a group, and all higher-numbered groups. The group number is 0 to 31.

All multi-byte integers are [big-endian](#).

4.6.5 SET GROUP LEVEL

44040000	group	ampl
----------	-------	------

The SET GROUP LEVEL command sets the amplitude level of a group.

The group number is 0 to 31.

Amplitude values (*ampl*) are expressed in units of 0.1 dB, with an offset of 1000. An amplitude value of 1000 is 0.0 dB; 900 is -10.0 dB. The maximum amplitude value at a group is 1100, which is +10.0 dB. An amplitude value of zero is off ($-\infty$ dB).

All multi-byte integers are [big-endian](#).

4.6.6 SET GROUP MUTE

5A040000	group	state
----------	-------	-------

The SET GROUP MUTE command turns on and off the mute state of a group.

State is one to turn mute on and zero to turn mute off.

All multi-byte integers are [big-endian](#).

4.6.7 SET GROUP SOLO

5B040000	group	state
----------	-------	-------

The SET GROUP SOLO command turns on and off the solo state of a group.

State is one to turn solo on and zero to turn solo off.

All multi-byte integers are [big-endian](#).

4.6.8 ZERO GROUPS

50040000

The ZERO GROUPS command sets the amplitude of all groups to zero ($-\infty$ dB).

All multi-byte integers are [big-endian](#).

4.7 EQ COMMANDS

An 8-band graphic equalizer exists on every input channel and every output channel, 128 in total, all identical in design. Each graphic equalizer is implemented as a cascade of 8 bi-quadratic sections, or ‘biquads’. The z-domain transfer function of a biquad is the ratio of two quadratic functions:

$$H(z) = \frac{a_2 + a_1z^{-1} + a_0z^{-2}}{1 + b_1z^{-1} + b_0z^{-2}}$$

Each biquad implements one band of equalization. Each band can be one of several filter types (Bell, Low Pass, High Pass, Low Shelf, High Shelf and Notch). Each band has a centre frequency, a positive or negative gain setting, and, if it is a Bell filter, a bandwidth setting. The TiMax user interface software supports the graphical design of 8-band filters, calculating the set of 5 coefficients for each biquad. The CPU writes these coefficients to the FPGA, setting the filter parameters. The FPGA implements the cascade of 8 biquads.

The Infinite Impulse Response (IIR) bi-quadratic sections are calculated using what is called Direct Form I. This method of calculating a biquad requires 5 multiply-accumulate (MAC) operations (yellow in Figure 2). Coefficients are a_0 , a_1 , a_2 , b_0 and b_1 (red in Figure 2). The incoming sample is x_n . The ‘n’ subscript is the index of the sample cycle. The output sample is the quantized result of the accumulation, y_n . The z^{-1} blocks (green) represent one sample-cycle delays. Thus when x_n is input to a z^{-1} block, the result is x_{n-1} , the input sample during the previous sample cycle. To implement the delays, five state variables ($x_n, x_{n-1}, y_n, y_{n-1}, e_n$) are kept in block RAM for each biquad.

To improve stability, a technique called first-order noise shaping is employed. After the summation is complete, the most-significant 48 bits of the result are split into two 24-bit values. The most-significant half is y_n and the least-significant half e_n is stored and added to the accumulator during the next sample cycle. This allows the biquad to perform nearly as well as a double-precision implementation at far lower implementation cost.

The function that calculates the coefficients is `EQGraph::calculateCoefficients`, based on the [Audio EQ Cookbook](#). Coefficients are provided as signed 32-bit numbers. This level of precision minimizes coefficient quantization problems which can affect fixed-point biquads at low frequencies. For the set of filter types supported by the TiMax2, all coefficient values (a_0 , a_1 , a_2 , b_0 , b_1) are between -8 and +8. For this range of values, in a 32-bit representation, the number “1” is 0x10000000. In this representation, -8 is 0x80000000 and +8 is 0x7FFFFFFF (actually $+8 - 2^{-32}$). A null biquad is one whose output always equals its input. The coefficients for a null biquad consists of a_2 equal to “1”, i.e. 0x10000000, and all other coefficients equal to zero. The values of the coefficients depend on the sample frequency, so when the sample frequency of a unit is changed, the coefficients must be re-calculated.

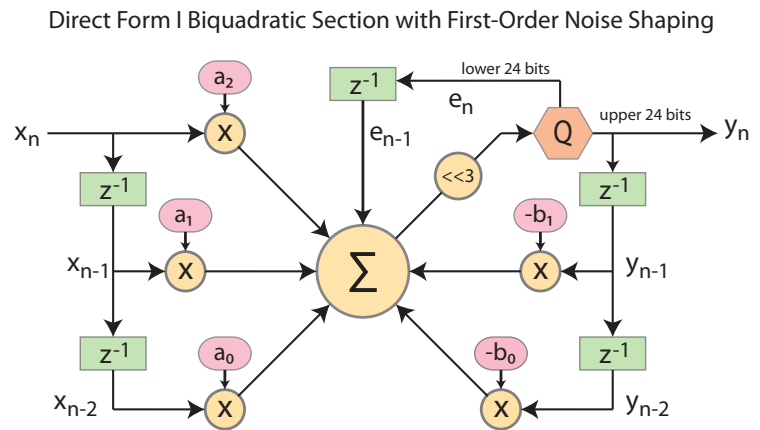


Figure 2: Direct Form I Biquad

4.7.1 CLEAR EQ CHANNELS

3F040000	io	start	end
----------	----	-------	-----

The CLEAR EQ CHANNELS command clears all EQ bands on a range of channels. The `io` parameter selects input channels when zero or output channels when one. The range of channels to clear is set by `start` to `end`, inclusive. The `start` and `end` channel numbers are 0 to 63 inclusive.

All multi-byte integers are [big-endian](#).

4.7.2 CLEAR EQ BANDS

190400cc	b1	...	bn	F0000000
----------	----	-----	----	----------

The CLEAR EQ BANDS command clears EQ bands on a channel.

The channel `cc`, 0 through 127 inclusive, is set in the lower byte of the [command code](#), with 0..63 for input and 64..127 for output.

A list of up to eight band numbers is included in the command. The list is terminated with a long constant as shown.

All multi-byte integers are [big-endian](#).

4.7.3 SET EQ

020400cc	band	ftype	0	gain	freq	bw
	A2	A1	A0	B1	B0	

The SET EQ command sets the coefficients for one band of EQ on one channel.

The channel `cc`, 0 through 127 inclusive, is set in the lower byte of the [command code](#), with 0..63 for input and 64..127 for output.

There are eight filter bands available on each channel. `band` is 0 to 7 inclusive. Filter type `ftype` is defined by:

```
enum eFType { eFTypeBell = 0, eFTypeLowShelf, eFTypeHighShelf, eFTypeLowPass,
              eFTypeHighPass, eFTypeNotch, eFTypeBandpass};
```

The `gain` parameter is expressed as $dB \times 10$. For example, if the filter gain should be +14.6dB, the parameter is 146. The `freq` parameter is expressed as $f_c \times 100$. For example, if the filter centre frequency should be 225.75 Hz, the parameter is 22575. The `bw` parameter is expressed as $bw_{oct} \times 1000$. For example, if the filter bandwidth should be one octave, the parameter is 1000. The range for filter bandwidth is 300 to 3000 inclusive.

The coefficients `A2`, `A1`, `A0`, `B1`, `B0` are calculated based on the above parameters. The function that calculates the coefficients is `EQGraph::calculateCoefficients..`

All multi-byte integers are [big-endian](#).

4.8 SHOW CONTROLLER COMMANDS

4.8.1 GO CUE

03040000	cn	s1	s2	ms
----------	----	----	----	----

The GO CUE command triggers the execution of a cue in the show currently open on the TiMax2 unit.

A cue number consists of three parts, the major number `cn` and one or two optional sub-numbers, `s1` and `s2`. In software, a complete cue number is displayed, for example, like this: 316.12.2. The major number is between 0 and 65532. Sub-numbers are between 1 and 255. If the sub-numbers are unused, they are set to zero.

The `ms` value sets the time within the cue where execution starts. It is typically set to zero.

All multi-byte integers are [big-endian](#).

4.8.2 OPEN SHOW

98040000	fileNumber
----------	------------

The OPEN SHOW command opens a show file in the TiMax2 show controller. The directory entry for the file (see [GET DIRECTORY](#)) contains the `fileNumber` required by this command.

When this command is executed, the previously-open show, if any, is closed, and the specified show is opened. Cues and image definitions in the new show are then available for recall.

All multi-byte integers are [big-endian](#).

4.8.3 SET CUE CLOCK

06040000	cn	s1	s2	ms
----------	----	----	----	----

The SET CUE CLOCK command stops the cue clock, sets the current cue number, sets the cue clock value and stops all live cues. This command is used when the playhead is manually set in the TiMax2 timeline window.

A cue number consists of three parts, the major number `cn` and one or two optional sub-numbers, `s1` and `s2`. In software, a complete cue number is displayed, for example, like this: 316.12.2. The major number is between 0 and 65532. Sub-numbers are between 1 and 255. If the sub-numbers are unused, they are set to zero.

The `ms` parameter is the millisecond count to which the clock is set.

All multi-byte integers are [big-endian](#).

4.8.4 STOP ALL

62040000

The STOP ALL command stops all live cues, all ongoing amplitude and delay ramps and all playback.

All multi-byte integers are [big-endian](#).

4.8.5 STOP ALL CUES

28040000

The STOP ALL CUES command clears all live commands and stops the cue clock. Live cues are cues which have been triggered and have not yet completed. There are zero live cues after this command is executed. Note that this command does not stop amplitude and delay ramps that have already been started by a cue and are still underway. Nor does it stop playback that was started by a cue. See the STOP RAMPS and the STOP ALL commands.

All multi-byte integers are [big-endian](#).

4.8.6 STOP CUE

04040000	cn	s1	s2
----------	----	----	----

The STOP CUE command stops the specified cue. Note that this command does not stop amplitude and delay ramps that have already been started by the cue and are still underway. Nor does it stop playback that was started by the cue. See the STOP RAMPS and the STOP ALL commands.

A cue number consists of three parts, the major number `cn` and one or two optional sub-numbers, `s1` and `s2`. In software, a complete cue number is displayed, for example, like this: 316.12.2. The major number is between 0 and 65532. Sub-numbers are between 1 and 255. If the sub-numbers are unused, they are set to zero.

All multi-byte integers are [big-endian](#).

4.8.7 STOP RAMPS

27040000

The `STOP RAMPS` command stops all ongoing amplitude and delay ramps.

All multi-byte integers are [big-endian](#).

4.8.8 RUN MIDI TIMECODE GENERATOR

2C040000	rate	h	m	s	f
----------	------	---	---	---	---

`RUN MIDI TIMECODE GENERATOR` sets the MIDI Timecode Generator frame rate and clock time, and starts the MIDI Timecode Generator. When running, the MIDI Timecode Generator transmits timecode on MIDI port 2 at quarter-frame intervals. The `rate` parameter is 0 for 24 frames per second, 1 for 25 frames per second, 2 for 30 drop frames per second and 3 for 30 frames per second. The `h` parameter contains hours, 0 through 23 inclusive. The `m` parameter contains minutes, 0 through 59 inclusive. The `s` parameter contains seconds, 0 through 59 inclusive. The `f` parameter contains frames, 0 through 29 inclusive; the maximum value depends on current frame-rate setting.

All multi-byte integers are [big-endian](#).

4.8.9 SET MIDI TIMECODE GENERATOR

29040000	h	m	s	f
----------	---	---	---	---

`SET MIDI TIMECODE GENERATOR` sets the MIDI Timecode Generator clock time. The `h` parameter contains hours, 0 through 23 inclusive. The `m` parameter contains minutes, 0 through 59 inclusive. The `s` parameter contains seconds, 0 through 59 inclusive. The `f` parameter contains frames, 0 through 30 inclusive; the maximum value depends on current frame-rate setting.

All multi-byte integers are [big-endian](#).

4.8.10 START MIDI TIMECODE GENERATOR

2B040000

`START MIDI TIMECODE GENERATOR` starts the MIDI Timecode Generator. When running, the MIDI Timecode Generator transmits timecode on MIDI port 2 at quarter-frame intervals.

All multi-byte integers are [big-endian](#).

4.8.11 STOP MIDI TIMECODE GENERATOR

2D040000

`STOP MIDI TIMECODE GENERATOR` stops the MIDI Timecode Generator.

All multi-byte integers are [big-endian](#).

4.8.12 RUN SHOW CLOCK

31040000	ms
----------	----

`RUN SHOW CLOCK` sets the show clock time, and starts the show clock. The `ms` parameter is the millisecond count to which the clock is set. The current show clock value, in milliseconds, is returned in the [unitStats structure](#).

All multi-byte integers are [big-endian](#).

4.8.13 SET SHOW CLOCK

2E040000	ms
----------	----

SET SHOW CLOCK sets the show clock time. The ms parameter is the millisecond count to which the clock is set. The current show clock value, in milliseconds, is returned in the [unitStats structure](#).

All multi-byte integers are [big-endian](#).

4.8.14 START SHOW CLOCK

2F040000

START SHOW CLOCK starts the show clock time. The current show clock value, in milliseconds, is returned in the [unitStats structure](#).

All multi-byte integers are [big-endian](#).

4.8.15 STOP SHOW CLOCK

30040000

STOP SHOW CLOCK stops the show clock time. The current show clock value, in milliseconds, is returned in the [unitStats structure](#).

All multi-byte integers are [big-endian](#).

4.9 PLAYBACK COMMANDS

4.9.1 PLAYBACK LOAD

200400cc	start	n0	...	nn	00000000
----------	-------	----	-----	----	----------

The `PLAYBACK LOAD` command loads an audio file, at a specified start location within the file, onto an input channel for playback. The input channel `cc` is set in the lower byte of the [command code](#).

The `start` location is specified in blocks. The TiMax2 playback system locates audio at block boundaries. A block is 5.333 milliseconds. If `start` is set to zero, playback will be at the beginning of the audio file.

The audio file name is included as a UTF-8 string. The first character of the name string is shown as `n0` and the last character of the string is shown as `nn`. The string is terminated by a zero long integer. The maximum string length (before termination long integer) is 60 bytes.

After a `PLAYBACK LOAD` command, the input channel is loaded at the start location for playback. A `RESUME PLAYBACK` command that includes the input channel will start playback.

(The [PLAYBACK GO](#) command loads an audio file on a channel and begins playback as soon as the load is completed.)

All multi-byte integers are [big-endian](#).

4.9.2 PLAYBACK LOAD NUMBER

250400cc	start	fileNumber
----------	-------	------------

The `PLAYBACK LOAD NUMBER` command loads an audio file, at a specified start location within the file, onto an input channel for playback. The file is specified by file number rather than name. The input channel `cc` is set in the lower byte of the [command code](#).

The `start` location is specified in blocks. The TiMax2 playback system locates audio at block boundaries. A block is 5.333 milliseconds. If `start` is set to zero, playback will be at the beginning of the audio file.

The directory entry for the file (see [GET DIRECTORY](#)) contains the `fileNumber` required by this command.

After a `PLAYBACK LOAD NUMBER` command, the input channel is loaded at the start location for playback. A `RESUME PLAYBACK` or `PLAYBACK GO SET` command that includes the input channel will start playback.

(The [PLAYBACK GO](#) command loads an audio file on a channel and begins playback as soon as the load is completed.)

All multi-byte integers are [big-endian](#).

4.9.3 PLAYBACK LOAD LOOP

0F0400cc	start	stop	loopCount	xfade	loopFrom	loopTo	n0	...	nn	00000000
----------	-------	------	-----------	-------	----------	--------	----	-----	----	----------

The `PLAYBACK LOAD LOOP` command loads an audio file with looping, at a specified start location within the file, onto an input channel for playback. The input channel `cc` is set in the lower byte of the [command code](#).

The `start` location is specified in blocks. The TiMax2 playback system locates audio at block boundaries. A block is 5.333 milliseconds. If `start` is set to zero, playback will be at the beginning of the audio file.

An audio loop is programmed into the audio file. When playback reaches the `loopFrom` location, playback jumps to the `loopTo` location. The loop will occur `loopCount` times, after which playback will continue normally, rather than jump, at the `loopFrom` location. If `loopCount` is specified as `0x3FFF`, looping will continue to occur indefinitely.

A crossfade smooths the audio at the jump location. The crossfade time `xfade` is specified in milliseconds. The minimum crossfade time is 5 milliseconds and the maximum is 85 milliseconds.

The audio file name is included as a UTF-8 string. The first character of the name string is shown as `n0` and the last character of the string is shown as `nn`. The maximum name string length (before termination long integer) is 60 bytes. The string is terminated by a zero long integer.

After a `PLAYBACK LOAD LOOP` command, the input channel is loaded at the start location for playback. A `RESUME PLAYBACK` or `PLAYBACK GO SET` command that includes the input channel will start playback.

(The [PLAYBACK GO](#) command loads an audio file on a channel and begins playback as soon as the load is completed.)

All multi-byte integers are [big-endian](#).

4.9.4 PLAYBACK GO

210100cc	start	n0	...	nn	00000000
----------	-------	----	-----	----	----------

The `PLAYBACK GO` command loads an audio file, at a specified start location within the file, onto an input channel, and when the audio buffer for the channel is filled to a minimum level, playback begins. The input channel `cc` is set in the lower byte of the [command code](#).

The `start` location is specified in blocks. The TiMax2 playback system locates audio at block boundaries. A block is 5.333 milliseconds. If `start` is set to zero, playback will be at the beginning of the audio file.

The audio file name is included as a UTF-8 string. The first character of the name string is shown as `n0` and the last character of the string is shown as `nn`. The maximum name string length (before termination long integer) is 60 bytes. The string is terminated by a zero long integer.

All multi-byte integers are [big-endian](#).

4.9.5 PLAYBACK GO LOOP

150100cc	start	stop	loopCount	xfade	loopFrom	loopTo	n0	...	nn	00000000
----------	-------	------	-----------	-------	----------	--------	----	-----	----	----------

The `PLAYBACK GO LOOP` command loads an audio file with looping, at a specified start location within the file, onto an input channel, and when the audio buffer for the channel is filled to a minimum level, playback begins. The input channel `cc` is set in the lower byte of the [command code](#).

The `start` location is specified in blocks. The TiMax2 playback system locates audio at block boundaries. A block is 5.333 milliseconds. If `start` is set to zero, playback will be at the beginning of the audio file.

An audio loop is programmed into the audio file. When playback reaches the `loopFrom` location, playback jumps to the `loopTo` location. The loop will occur `loopCount` times, after which playback will continue normally, rather than jump, at the `loopFrom` location. If `loopCount` is specified as `0x3FFF`, looping will continue to occur indefinitely.

A crossfade smooths the audio at the jump location. The crossfade time `xfade` is specified in milliseconds. The minimum crossfade time is 5 milliseconds and the maximum is 85 milliseconds.

The audio file name is included as a UTF-8 string. The first character of the name string is shown as `n0` and the last character of the string is shown as `nn`. The maximum name string length (before termination long integer) is 60 bytes. The string is terminated by a zero long integer.

All multi-byte integers are [big-endian](#).

4.9.6 PLAYBACK GO SET

11040000	f1	f2
----------	----	----

The `PLAYBACK GO SET` command starts playback on a set of input channels. For TiMax2, the set is defined by two long integer flags, `f1` and `f2`. The bits in `f1` corresponds to input channels 0 through 31, inclusive. The bits in `f2` corresponds to input channels 32 through 63, inclusive.

Playback is started on channels for which the bit is set and that are playback loaded, i.e. have a playback file and location assigned.

All multi-byte integers are [big-endian](#).

4.9.7 PLAYBACK PAUSE

23040000	low	high
----------	-----	------

The `PLAYBACK PAUSE` command stops playback on a range of input channels. The range is set by `low` through `high` inclusive. If a channel is not loaded, i.e. does not have a playback file and location assigned, nothing happens. Paused playback channels are still loaded with the playback file that was playing, and can be restarted from the paused location with a `PLAYBACK RESUME` command.

All multi-byte integers are [big-endian](#).

4.9.8 PLAYBACK QUEUE

330400cc	start	n0	...	nn	00000000
----------	-------	----	-----	----	----------

The `PLAYBACK QUEUE` command queues an audio file onto an input channel. The input channel `cc` is set in the lower byte of the [command code](#).

An input channel can have any number of audio files queued for playback. When playback on the current audio file completes, the next file in the queue is started at the `start` location specified.

The `start` location is specified in blocks. The TiMax2 playback system locates audio at block boundaries. A block is 5.333 milliseconds. If `start` is set to zero, playback will be at the beginning of the audio file.

The audio file name is included as a UTF-8 string. The first character of the name string is shown as `n0` and the last character of the string is shown as `nn`. The maximum name string length (before termination long integer) is 60 bytes. The string is terminated by a zero long integer.

All multi-byte integers are [big-endian](#).

4.9.9 PLAYBACK RESUME

22040000	low	high
----------	-----	------

The `PLAYBACK RESUME` command starts playback on a range of input channels. The range is set by `low` through `high` inclusive. Playback is started on channels in the specified range that are playback loaded, i.e. have a playback file and location assigned.

All multi-byte integers are [big-endian](#).

4.9.10 PLAYBACK STOP LOOPING

4F0400cc

The `PLAYBACK STOP LOOPING` command stops playback looping on an input channel. The input channel `cc` is set in the lower byte of the [command code](#). If playback is looping on an input channel, after this command is received on that channel, playback will continue normally, rather than jump, at the `loopFrom` location.

All multi-byte integers are [big-endian](#).

4.9.11 PLAYBACK UNLOAD

24040000	low	high
----------	-----	------

The `PLAYBACK UNLOAD` command unloads a range of input channels. The range is set by `low` through `high` inclusive. Channels in the specified range are unloaded, i.e. no longer have a playback file and location assigned.

All multi-byte integers are [big-endian](#).

4.10 GET STATUS COMMANDS

4.10.1 AUTOUPDATE

85060000	en	all
----------	----	-----

An AUTOUPDATE command activates the TiMax automatic status updating system. Up to four computer systems can be connected to a single TiMax for autoupdate.

Rather than polling the TiMax unit for status data by periodically sending the commands in this section, it is possible to automate much of this by asking instead for the unit to return data only when data has changed. This reduces the ethernet bandwidth considerably, particularly when the system is relatively idle.

It also makes the coding of a software system that displays TiMax status information simpler. Such a software system will have local copies of the data structures shown in this section, `struct unitXptDelay`, `struct unitXptLevels`, etc. When updated versions of these structures are received, the local copies are overwritten. (On incoming data, it is necessary to translate the [endian-ness](#) of the multi-byte integers.) This way, the local copies are always up-to-date.

When such a software system boots, it needs to receive all data structures to update its local copies. This is done by sending an AUTOUPDATE command with the `all` parameter set to 1. Every data structure in the autoupdate system is sent to the software system.

After this, AUTOUPDATE commands are sent with the `all` parameter set to 0. This requests the TiMax unit to send updates for all data structures that have changed since the last time an AUTOUPDATE command was seen from that software system. A single AUTOUPDATE command can be sent periodically, at frame rate for example, to keep the software up-to-date.

The TiMax unit needs to see an AUTOUPDATE command periodically. Otherwise, the unit times out that autoupdate connection for that computer system. The timeout period is 30 seconds.

The autoupdate enable flag `en` should be 1 every time an AUTOUPDATE command is sent, except when the software is pausing or shutting down. In this case, an AUTOUPDATE command with the enable flag `en` equal to 0 will immediately close the autoupdate connection for that computer system.

All of the GET XXX commands in this section (below) are in the autoupdate system, except GET DIRECTORY.

The directory only changes when a file is downloaded or erased. Audio and show file download are reserved for the TiMax software. A directory upload is potentially quite large, and software should do a GET DIRECTORY command infrequently.

4.10.2 GET CROSSPOINT DELAYS

72060000	startCh
----------	---------

The GET CROSSPOINT DELAYS command requests the TiMax unit to return a UDP packet containing a `unitXptDelay` structure. This structure contains the current delays for 512 crosspoints. The `startCh` parameter specifies the first output channel in the set of crosspoints contained in the returned packet.

The `startCh` parameter is one of: 0, 8, 16, 24, 32, 40, 48, 56. It is necessary to perform eight GET CROSSPOINT DELAYS commands to retrieve the current delays for a 64-channel matrix.

If the unit is configured with 128 output channels, output channel numbers divisible by four are valid. When `startCh` is zero, for example, the packet returned by the GET CROSSPOINT DELAYS command contains the crosspoint delays for output channels 0 through 3. It is necessary to perform 64 GET CROSSPOINT DELAYS commands to retrieve the current delays for a 128-output-channel matrix.

Delay values are expressed in units of 0.1 mS. The maximum delay value is 9999, which is 999.9 mS, just under one second.

```
struct unitXptDelay {
    int code;          //72060000
    int startOutChannel;
    short xptDelay[8][64]; // [output][input] in dlyn
};
```

All multi-byte integers are [big-endian](#).

4.10.3 GET CROSSPOINT LEVELS

71060000	startCh
----------	---------

The GET CROSSPOINT LEVELS command requests the TiMax2 unit to return a UDP packet containing a `unitXptLevels` structure. This structure contains the current amplitude levels for 512 crosspoints. The `startCh` parameter specifies the first output channel in the set of crosspoints contained in the returned packet. The `startCh` parameter is one of: 0, 8, 16, 24, 32, 40, 48, 56. It is necessary to perform eight GET CROSSPOINT LEVELS commands to retrieve the current crosspoint levels for a 64-channel matrix.

```
struct unitXptLevels {
    int code;          //71060000
    int startOutChannel;
    short xptLev[8][64]; // [output][input] in dBn
};
```

If the unit is configured with 128 output channels, output channel numbers divisible by four are valid. When `startCh` is zero, for example, the packet returned by the GET CROSSPOINT LEVELS command contains the crosspoint levels for output channels 0 through 3. It is necessary to perform 64 GET CROSSPOINT LEVELS commands to retrieve the current crosspoint levels for a 128-output-channel matrix.

```
struct unitXptLevels {
    int code;          //71060000
    int startOutChannel;
    short xptLev[512]; // in dBn
};
```

Amplitude values are expressed in units of 0.1 dB, with an offset of 1000. An amplitude value of 1000 is 0.0 dB; 900 is -10.0 dB. The maximum amplitude value at a group is 1100, which is +10.0 dB. An amplitude value of zero is off ($-\infty$ dB).

All multi-byte integers are [big-endian](#).

4.10.4 GET DIRECTORY

81060000

The GET DIRECTORY command requests the TiMax unit to return a set of packets specifying the contents of the show file directory and the audio file directory. Each returned packet contains one `unitDirectory` structure.

One `struct directoryEntry` is returned for every file. The `struct directoryEntry` includes the file name string in UTF-8 format. Up to eight directory entries are sent in each ethernet packet. A series of ethernet packets is sent, until all directory entries have been sent. The last packet in the series is marked with `lastPacket` equal to one. The `fileType` parameter in each directory entry is zero for audio files and one for show files.

```
struct directoryEntry {
    int dirfileNumber;
    char fileType;
    char channels;
    char sampleRate;
    char loopCrossfades;
    int blocks;
    int startBlock;
    int blocks2;
    int startBlock2;
    int timestamp;
    int unused2; //used as link in firmware
    char filename[64];
    char unused3[32]; //for future use
};
struct unitDirectory {
    int code;          //81060000
    int firstFileNumber; //first file number in this packet
```

```
int entriesInPacket; //unused
int lastPacket;
struct directoryEntry entry[8];
```

4.10.5 GET DISK STATE

77060000

The GET DISK STATE command requests the TiMax unit to return a packet containing a `unitDiskState` structure. This structure contains a variety of data, including data about the internal disk drive.

```
struct unitDiskInfo {
    int capacity;           //disk capacity in blocks
    int diskBlksInUse;     //disk blocks in use
    int initError;
    char model[40];        //disk model string
    int unused6;
    char serialNumber[20]; //disk serial number string
};

struct unitDiskState {
    int code; //77060000
    int diskCapacity; //total capacity of the disk in blocks
    int highestSelNum; //highest selection number in use
    int delayAllocation; //xpt delay line length (unused- see Mixer::setInputsWithDelay)
    char transferActive; //audio or show file transfer is in progress (unused)
    char GPOstate; //state of GPO outputs, bit 0: GPO 1, bit 1: GPO 2
    char autoConfIP; //using auto-configured IP address
    char disksReady; //bit 0: main ready, bit 1: backup ready, b2: backup active
    char MIDITestResults; //boot MIDI loopback test results: bit 0: port 1, bit 1: port 2
    uchar midiCh;
    char unused1[2]; //unused
    int defaultShow; //default show number
    int fixedIPtag; //fixed IP tag
    int fixedIP; //fixed IP
    int GPIplbkTrigMode; //GPI playback trigger mode
    int GPItrigLockout; //GPI trigger lockout time in milliseconds
    struct unitDiskInfo diskinfo;
};
```

All multi-byte integers are [big-endian](#).

4.10.6 GET IO LEVELS

68060000

The GET IO LEVELS command requests the TiMax unit to return a packet containing a `unitIOLevels` structure. This structure contains the current amplitude settings for the source mix, input levels, output levels and group levels, as well as other data.

Amplitude values are expressed in units of 0.1 dB, with an offset of 1000. An amplitude value of 1000 is 0.0 dB; 900 is -10.0 dB. The maximum amplitude value at a group is 1100, which is +10.0 dB. An amplitude value of zero is off ($-\infty$ dB).

The following structure is returned:

```
struct unitIOLevels {
    int code; //68060000
    short ioCardSrcLev[64];
    short moduleSrcLev[64];
    short playbackSrcLev[64];
    short IOLev[128];
    short recalledXptRow[64];
    int groupLevels[32];
    int recallRowRampTime;
```

```

char recallRowGainEnable;
char recallRowDelayEnable;
char unused1[2];
};

```

All multi-byte integers are [big-endian](#).

4.10.7 GET EQ

78060000	startCh
----------	---------

The GET EQ command requests the TiMax unit to return a packet containing a `unitEQ` structure. This structure contains the current EQ parameters for eight channels. There are eight EQ bands per channel.

The `startCh` parameter is one of: 0, 8, 16, 24, 32, 40, 48, 56 for input channels and 64, 72, 80, 88, 96, 104, 112, 120 for output channels. It is necessary to perform sixteen GET EQ commands to retrieve the current EQ parameters for a 64-channel matrix.

```

// filter types
enum eFType { eFTypeBell = 0, eFTypeLowShelf, eFTypeHighShelf, eFTypeLowPass,
              eFTypeHighPass, eFTypeNotch, eFTypeBandpass};
struct unitEQBand { //parameters for one band
    char filterType; // filter type, eFTypeLowShelf...
    char bypass; //unused
    short gain; //gain in dB * 10 (e.g. 146 for +14.6 dB)
    int frequency; // centre frequency * 100, 2000..2000000 (e.g. 22575 for 225.75 Hz)
    int bw; //bandwidth in octaves * 1000, 300..3000
};
struct unitEQChannel { //parameters for one channel
    struct unitEQBand band[8];
};
struct unitEQ {
    int code; //78060000
    int startChan; //starting channel number: 0, 8, 16...120
    struct unitEQChannel ueq[8];
};

```

All multi-byte integers are [big-endian](#).

4.10.8 GET GROUP ASSIGNS

76060000

The GET GROUP ASSIGNS command requests the TiMax unit to return a packet containing a `groupAssigns` structure. This structure contains the current channel assignments for all groups.

In TiMax2, for each group, 16 bytes are returned. These bytes contain 128 flags, one for each channel, numbered 0 to 63 for input channels and 64 to 127 for output channels. If a channel is assigned to a group, the flag is set. Phase flags for all channels are also returned.

```

struct groupAssigns {
    int code; //76060000
    unsigned char assigns[32][16]; //current group assigns
    unsigned char phaseFlags[128]; //current phase flags
};

struct groupAssigns {
    int code; //ccGetGroupAssigns

```

```
unsigned char assigns[16][2][32]; //current group assigns, groups 0 to 15, 0: inputs, 1: outputs
unsigned char inPhaseFlags[256]; //current phase flags
};
struct upperGroupAssigns {
    int code; //ccGetUpperGroupAssigns
    unsigned char assigns[16][2][32]; // //current group assigns, groups 16 to 31, 0: inputs, 1: outputs
    unsigned char outPhaseFlags[256]; //current phase flags
};
```

All multi-byte integers are [big-endian](#).

4.10.9 GET METERING

70060000

The GET METERING command requests the TiMax2 unit to return a packet containing a `unitMetering` structure. This structure contains the current metering levels for all input and output channels and reverb engines. For input and output channels, the returned value ranges from 0 to 122, with values above 122 representing clipping.

For the reverb engines, a 24-bit metering value is returned.

```
struct unitMetering {
    int code;          //70060000
    unsigned char meter[128];
    unsigned int revMeter[4];
};
```

All multi-byte integers are [big-endian](#).

4.10.10 GET MODULE

67060000

The GET MODULE command requests the TiMax2 unit to return a packet containing a `moduleStruct` structure. This structure contains a variety of data, including data about the installed network audio modules..

```
struct moduleStruct {
    int code; //67060000
    short CN1RXBundle[4]; //Module 1 receiver 0..3 bundle assignments
    short CN1TXBundle[4]; //Module 1 transmitter 0..3 bundle assignments
    short CN2RXBundle[4]; //Module 2 receiver 0..3 bundle assignments
    short CN2TXBundle[4]; //Module 2 transmitter 0..3 bundle assignments
    int CPUlogic;
    int LCDlogic;
    int DSP0logic;
    int DSP1logic;
    int DSP2logic;
    int DSP3logic;
    int IO0logic;
    int IO1logic;
    int IO2logic;
    int IO3logic;
    short module1ID;
    unsigned char module1VMajor;
    unsigned char module1VMinor;
    short module2ID;
    unsigned char module2VMajor;
    unsigned char module2VMinor;
    unsigned char macAdrs[6];
    unsigned char productVersion[2]; //unused
    char firmwareVers[4];
    char ROMVers[4];
    int MADI1ControlRegister;
    int MADI2ControlRegister;
    int MADI1StatusRegister;
    int MADI2StatusRegister;
    char mtcOutFrameRate; //MTC generator frame rate (0..3)
    char FPGAinstalled; //FPGA detected
```

```

char FPGAmatrixSize; //FPGA assigned matrix size
char sampleFreq;     //0: 48KHZ, non-zero: 96kHz
char FPGAversion[88]; //FPGA version information
char reserved3[80];  //reserved for future use
char soloMode;
char reserved4[3];
int inputsWithDelay; //number of inputs with delay
char recXptRowEn;    //recall crosspoint row enabled
char plbkEn;        //playback enabled
char showTrigEn;    //external show triggers enabled
char reserved5;     //unused
int recRowRamp;     //recall row ramp time
char recRowGainEn;  //recall row ramp gain enable
char recRowDlyEn;   //recall row ramp delay enable
short devsetunused2; //unused
char aes3fpga[16];  //mio aes3 fpga data
};

```

All multi-byte integers are [big-endian](#).

4.10.11 GET MUTE SOLO

75060000

The GET MUTE SOLO command requests the TiMax2 unit to return a packet containing a `unitMuteSolo` structure. This structure contains the current states of the mutes and solos on the input channels, output channels and groups.

For the `ioMuteSolo` array, the mute state is in bit 0 of the returned byte and the solo state is in bit 1. The channel number is 0 to 127, with output channels starting at 64. For the `groupMute` `groupSolo` arrays, the state is in bit 0 of the returned byte.

```

struct unitMuteSolo {
    int code; //75060000
    char ioMuteSolo[128];
    char groupMute[32];
    char groupSolo[32];
};

```

All multi-byte integers are [big-endian](#).

4.10.12 GET OUTPUT DELAYS

69060000

The GET OUTPUT DELAYS command requests the TiMax2 unit to return a packet containing a `unitIODelay` structure. This structure contains the current delay values for output delays, in samples. There are 48 samples per millisecond when the sample rate is set to 48K; There are 96 samples per millisecond when the sample rate is set to 96K;

```

struct unitIODelay {
    int code; //69060000
    int inputDelay[64]; //unused
    int outputDelay[64]; //samples
};

```

All multi-byte integers are [big-endian](#).

4.10.13 GET PLAYBACK

73060000

The GET PLAYBACK command requests the TiMax2 unit to return a packet containing a unitPlayback structure. This structure contains the current playback status of all 64 playback channels.

```
struct unitPlayback {
    int code;          //73060000
    struct {
        char fileChannel;    //channel number in file (zero-based 0..n-1, n-channel file)
        char unused;
        char stopped;        //nonzero: channel is stopped
        char loaded;         //nonzero: audio is loaded on channel
        int selectionNumber; //selection number loaded on channel
        int playbackLocation; //playback location in blocks from start of audio file
        int audioTrackTime;  //timeline track time of start of audio file (<= startTime)
    } pch[64];
}
```

All multi-byte integers are [big-endian](#).

4.10.14 GET REVERB

6B060000

The GET REVERB command requests the TiMax2 unit to return a packet containing a reverb structure. This structure contains many current reverb parameters.

```
const int revParams = 5;
```

```
struct reverbParameters {
    char name[20];
    int mode;
    int futureUse;
    int paramMin[revParams];
    int paramMax[revParams];
};
```

```
struct reverb {
    int code; //6B060000
    ushort modc[revEngines]; //modulation table control words
    ushort preDelay[revEngines]; //pre-delay for each engine
    ushort FDNdelay[revEngines][revDlyLines]; //delay on FDN delay lines
    ushort postDelay[64]; //post-reverb delay
    uint coefficients[revEngines][revDlyLines][2]; //HPF/LPF filter coefficients
    ushort mtxFBGain[revEngines][revDlyLines]; //matrix feedback gain, dBn
    ushort dlyInGain[revEngines][revDlyLines]; //delay line-in gain, dBn
    ushort preDlyFBGain[revEngines]; //pre-delay feedback gains, dBn
    uint prePostFaderFlags[revEngines][2]; //pre-post fader flags
    ushort preMixGain[revEngines]; //pre-mix gain
    reverbParameters preset[revEngines]; //preset parameters
};
```

All multi-byte integers are [big-endian](#).

4.10.15 GET REVERB EQ PARAM

65060000

The GET REVERB EQ PARAM command requests the TiMax2 unit to return a packet containing a reverbParam structure. This structure contains current reverb EQ parameter settings.

```
// filter types
enum eFType { eFTypeBell = 0, eFTypeLowShelf, eFTypeHighShelf, eFTypeLowPass,
  eFTypeHighPass, eFTypeNotch, eFTypeBandpass};
struct unitEQBand { //parameters for one band
  char filterType; // filter type, eFTypeLowShelf...
  char bypass; //unused
  short gain; //gain in dB * 10 (e.g. 146 for +14.6 dB)
  int frequency; // centre frequency * 100, 2000..2000000 (e.g. 22575 for 225.75 Hz)
  int bw; //bandwidth in octaves * 1000, 300..3000
};
struct reverbParam {
  int code; //65060000
  struct {
    struct unitEQBand param;
    uint coeff[5];
  } band[4][8]; //engine number 0..3, band number 0..7
};
```

All multi-byte integers are [big-endian](#).

4.10.16 GET REVERB LEVELS

6A060000

The GET REVERB LEVELS command requests the TiMax2 unit to return a packet containing a reverbLevels structure. This structure contains current reverb input and matrix levels.

```
struct reverbLevels {
  int code; //6A060000
  short inputLev[4][64];
  short matrixLev[4][64];
};
```

All multi-byte integers are [big-endian](#).

4.10.17 GET REVERB MOD TABLE

57060000 index

The GET REVERB MOD TABLE command requests the TiMax2 unit to return a packet containing a reverbModTable structure. This structure contains 1024 bytes of the current reverb modulation table. The index parameter specifies the starting byte index in the reverb modulation table. index is one of: 0, 1024, 2048, 3072.

```
struct reverbModTable {
  int code; //57060000
  int index; //byte index where this data goes in modulation table
  uint page[1024];
};
```

All multi-byte integers are [big-endian](#).

4.10.18 GET REVERB OUTPUT GAIN

89060000

The GET REVERB OUTPUT GAIN command requests the TiMax2 unit to return a packet containing a `reverbOutGain` structure. This structure contains current reverb output gain settings.

```
struct reverbOutGain {
    int code;    //89060000
    int index;   //byte index where this data goes in total output gain array
    ushort page[8][4][16]; //channel, engine, delayLine
};
```

All multi-byte integers are [big-endian](#).

4.10.19 GET STATUS

6F060000

The GET STATUS command requests the TiMax2 unit to return a packet containing a `unitStats` structure. This structure contains an assortment of status indicators, many of which are intended for diagnostic purposes.

```
struct unitStats {
    int code;           //75060000
    int freeBuffers;    //number of entries on the free queue
    int lowWaterMark;   //free queue low water mark
    int timerCount;     //number of entries on the timer queue
    int workCount;      //number of entries on the work queue
    int MIDI1TXCount;   //number of entries on the MIDI 1 transmit queue
    int MIDI2TXCount;   //number of entries on the MIDI 2 transmit queue
    int currentTime;    //current millisecond count
    int showClock;      //show clock (ms)
    int showMemoryMax;  //maximum amount of memory used by current show
    int cpuUtilization; //CPU utilization percent
    int enetMsgs;       //messages received via ethernet
    int enetOKMsgs;     //valid messages received via ethernet
    int enetBADMsgs;    //invalid messages received via ethernet
    int MIDI1rxMsgs;    //messages received via MIDI port 1
    int MIDI1OKMsgs;    //valid messages received via MIDI port 1
    int MIDI1BADMsgs;   //invalid messages received via MIDI port 1
    int MIDI2rxMsgs;    //messages received via MIDI port 2
    int MIDI2OKMsgs;    //valid messages received via MIDI port 2
    int MIDI2BADMsgs;   //invalid messages received via MIDI port 2
    int MIDI1txMsgs;    //messages transmitted via MIDI port 1
    int MIDI2txMsgs;    //messages transmitted via MIDI port 2
    int temperature;    //internal unit temperature
    int taskNoBufs;     //task no buffer errors
    int showQueueEntries; //entries in the show queue
    int plbkChannelsActive; //channels that are playing back
    int stallBlks;      //audio blocks not sent waiting for HDD
    char cueClkRunning; //current cue clock running
    char showClkRunning; //show clock running
    char graphicFileInShow; //show contains map graphic file
    char currentSampleRate; //current sample rate, set when unit boots
    int drqCount;        //count of entries in disk request queue
    int freeDRPBcount;   //number of free DRPBs
```

```

int freeDRPBlwm;           //number of free DRPBs low water mark
int badDMrequests;        //count of bad disk manager requests
int GPIport;              //current value of GPI port pins
int bootupDelay;         //boot up delay, seconds
int diskReads;           //disk read count
int diskWrites;          //disk write count
int enetRxCount;         //ethernet packets received
int enetTxCount;         //ethernet packets transmitted
int enetNoBufs;          //ethernet no buffer errors
int enetRXovrs;          //ethernet receive overrun errors
int enetTxBufallocs;     //ethernet transmit buffer allocations
int enetCRCerrors;       //ethernet receive CRC errors
int enetChecksumErrors; //ethernet receive checksum errors
int zeroConfCount;       //zero configuration attempts
int enetNonOctetErrors; //ethernet receive non-octet errors
int backupFileNumber;    //backup file number
int enetOversizePackets; //ethernet receive oversize packet errors
int cueClock;            //current cue clock - starts at 0 when cue is triggered
int enetTruncatedPacketErrors; //ethernet receive truncated packet errors
int showFileBadDisk;     //count of show file open errors
int liveCmds;            //count of live commands
char rtString[23];        //real time string
char rtBatteryLow;        //real time clock battery low
char MTCINrunning;       //incoming MTC running
char MTCINframeRate;     //incoming MTC frame rate (0..3)
char MTCINhours;         //incoming MTC hours
char MTCINminutes;       //incoming MTC minutes
char MTCINseconds;       //incoming MTC seconds
char MTCINframes;        //incoming MTC frames
char MTCGrunning;        //MTC Generator running
char MTCGframeRate;      //MTC Generator frame rate (0..3)
char MTCGhours;          //MTC Generator hours
char MTCGminutes;        //MTC Generator minutes
char MTCGseconds;        //MTC Generator seconds
char MTCGframes;         //MTC Generator frames
int showNumber; //open show number (0 = none open)
uint currentCueNumber; //number of last cue to go
int liveCues;            //number of live cues
int dhcpLease;           //DHCP lease time in seconds
};

```

All multi-byte integers are [big-endian](#).

4.10.20 GET TRACKING ENABLES

66060000

The GET TRACKING ENABLES command requests the TiMax2 unit to return a packet containing a `unitTracking` structure. This structure contains the current states of the tracking enable flags on the 64 input channels. A channel has tracking enabled if the `trackingEnabled` byte is non-zero.

```

struct unitTracking {
    int code;           //66060000
    char trackingEnabled[64];
};

```

All multi-byte integers are [big-endian](#).

4.10.21 INQUIRY

6EA70004

Software can discover TiMax units on its local area network by broadcasting an INQUIRY command to the TiMax port (0xE7C3). An INQUIRY command consists of a single long integer command code. All TiMax units which are reachable on the local network will reply to INQUIRY with a packet that contains the IP address of the unit, as well as other unit information. After receiving this reply, the software thereafter transmits all packets directly to each TiMax unit using the unit's IP address. Only the INQUIRY packet is ever broadcast. The software broadcasts an inquiry packet periodically, so it knows when another TiMax unit joins the network.

If a local area network does not support broadcast messages, the INQUIRY method does not work. In this case, the unit should be set to a fixed IP address and the connection made via the software's "Connect at IP" command (in the "Unit" menu). Once the fixed IP is entered, the software can connect directly with the TiMax unit at that address without having to broadcast an inquiry packet.

A TiMax unit that receives a INQUIRY command responds by sending a packet containing an inquiry response:

```
struct unitInquiry {
    int code;    //ccInquiry
    uint IPaddress;    //IP address, in network byte order
    short inchannels;    //number of input channels (0, 16, 32, 48, 64, 128 or 256)
    short outchannels;    //number of output channels (0, 16, 32, 48, 64, 128 or 256)
    char mfg[8];
    char product[16];
    char version[4];
    char date[8];
    char time[8];
    char serialnum[8];
    char unitName[32];
    char mainDiskRDY;    //main disk ready flag
    char mainERR;    //main disk error number (unused)
    char backupDiskRDY;    //backup disk ready flag (unused)
    char backupDiskERR;    //backup disk error number (unused)
    char password[8];
    char fixedIPaddress;    //IP address is fixed
    char autoAssignedAddress;    //IP address was auto-assigned (no DHCP server)
    char key[20];
    char fpga;    //number of FPGAs: 0, 1 or 2
    char reqBroadcast;
    uint senderIPadrs;
    int showNumber;
    int showRevision;
    int XMLpackets;
    int showMemorySize;
    char showname[maxDirFileNameLen];
};
```

All multi-byte integers are [big-endian](#).

4.11 REVERB COMMANDS

4.11.1 SET REVERB DELAY LINE INPUT GAIN

C0040000	eng	delayLine	gain
----------	-----	-----------	------

The SET REVERB DELAY LINE INPUT GAIN command sets the input gain for one main FDN delay line on one reverb engine. The reverb engine `eng` is 0 to 3 inclusive. The delay line `delayLine` is 0 to 15 inclusive.

The amplitude value (`gain`) is expressed in units of 0.1 dB, with an offset of 1000. An amplitude value of 1000 is 0.0 dB; 900 is -10.0 dB. The maximum amplitude value at an input is 1100, which is +10.0 dB. An amplitude value of zero is off ($-\infty$ dB).

All multi-byte integers are [big-endian](#).

4.11.2 SET ALL REVERB DELAY LINE INPUT GAIN

CA040000	eng	gain0	...	gain15
----------	-----	-------	-----	--------

The SET ALL REVERB DELAY LINE INPUT GAIN command sets the input gain for all main FDN delay lines on one reverb engine. The reverb engine `eng` is 0 to 3 inclusive. The command contains a list of 16 `gain` parameters, which set the input gains on delay lines 0 to 15.

The amplitude value (`gain`) is expressed in units of 0.1 dB, with an offset of 1000. An amplitude value of 1000 is 0.0 dB; 900 is -10.0 dB. The maximum amplitude value at an input is 1100, which is +10.0 dB. An amplitude value of zero is off ($-\infty$ dB).

All multi-byte integers are [big-endian](#).

4.11.3 SET REVERB DELAY MODULATOR CONTROL

B8040000	eng	tableSize
----------	-----	-----------

The SET REVERB DELAY MODULATOR CONTROL command sets the table size for the FDN Delay Modulator on one reverb engine. Also enables or disables the FDN modulator.

The reverb engine `eng` is 0 to 3 inclusive. The table size `tableSize` is 1024 maximum.

To enable the modulator, set bit 10 of the `tableSize` long integer. To disable the modulator, clear bit 10 of the `tableSize` long integer.

All multi-byte integers are [big-endian](#).

4.11.4 SET REVERB DELAY MODULATOR ENTRIES

A9060000	eng	startIndex	e0	...	e255
----------	-----	------------	----	-----	------

The SET REVERB DELAY MODULATOR ENTRIES command sets 256 entries in the FDN Delay Modulator table on one reverb engine.

The reverb engine `eng` is 0 to 3 inclusive.

The starting table index `startIndex` is 0, 256, 512 or 768. Each entry `e0` through `e255` has four fields, defined as follows:

- `b8-b0` = Sample Block Count This is how many Block FS periods (8 x FS) the Modulator counts (and applies the modulation offset data in `b17-b9`) before reading the next table entry.
- `b11-9` = Sample Offset This value determines how many individual samples to offset the FDN delay line by, within the current offset sample block (see bits 16-12).
- `b16-b12` = Delay Block Offset This value determines how many 8 sample blocks to offset the FDN delay line by.
- `b17` = Delay offset sign/direction This bit determines whether the offset in bits 16-12 is applied to increase the FDN delay (logic '0') or decrease it (logic '1').

All multi-byte integers are [big-endian](#).

4.11.5 SET REVERB DELAY MODULATOR ENTRY

B9040000	eng	tableIndex	entry
----------	-----	------------	-------

The SET REVERB DELAY MODULATOR ENTRY command sets one entry in the FDN Delay Modulator table on one reverb engine.

The reverb engine `eng` is 0 to 3 inclusive.

The table index `tableIndex` is 0 to 1023 inclusive. The table entry `entry` is defined in [SET REVERB DELAY MODULATOR ENTRIES](#).

All multi-byte integers are [big-endian](#).

4.11.6 SET REVERB ENABLE

C5040000	enable
----------	--------

The SET REVERB ENABLE command enables or disables the reverb. The enable value `enable` is 0 to disable and 1 to enable.

All multi-byte integers are [big-endian](#).

4.11.7 SET REVERB FDN DELAY

BB040000	eng	delayLine	smpl
----------	-----	-----------	------

The SET REVERB FDN DELAY command sets the delay on one FDN delay line on one reverb engine. The reverb engine number `eng` is 0 to 3 inclusive. The delay line number `delayLine` is 0 to 15 inclusive. The `smpl` parameter is number of samples of delay divided by 8 (0 to 16383 inclusive).

Note that the current sample rate setting affects the translation between time and the `smpl` parameter. If the current sample rate in kHz is SR_{kHz} (either 48 or 96) and $delay_{ms}$ is the delay in milliseconds, the following equation applies:

$$smpl = \frac{delay_{ms} \times SR_{kHz}}{8}$$

All multi-byte integers are [big-endian](#).

4.11.8 SET ALL REVERB FDN DELAY

C7040000	eng	smpl0	...	smpl15
----------	-----	-------	-----	--------

The SET ALL REVERB FDN DELAY command sets the delay on all 16 FDN delay lines on one reverb engine. The reverb engine number `eng` is 0 to 3 inclusive. The `smpl` parameter is number of samples of delay divided by 8 (0 to 16383 inclusive). The command contains a list of 16 `smpl` parameters, which set the delays values on delay lines 0 to 15.

Note that the current sample rate setting affects the translation between time and the `smpl` parameter. If the current sample rate in kHz is SR_{kHz} (either 48 or 96) and $delay_{ms}$ is the delay in milliseconds, the following equation applies:

$$smpl = \frac{delay_{ms} \times SR_{kHz}}{8}$$

All multi-byte integers are [big-endian](#).

4.11.9 SET REVERB FDN FILTER COEFFICIENT

BD040000	eng	delayLine	ftype	coeff
----------	-----	-----------	-------	-------

The SET REVERB FDN FILTER COEFFICIENT command sets the highpass / lowpass filter coefficient for one FDN line on one reverb engine. The reverb engine number `eng` is 0 to 3 inclusive. The delay line number `delayLine` is 0 to 15 inclusive. The `ftype` parameter is 0 to instantiate a lowpass filter and 1 to instantiate a highpass filter. The two's-complement 32-bit coefficient data is scaled so that $0x40000000 = +0.5$.

All multi-byte integers are [big-endian](#).

4.11.10 SET ALL REVERB FDN FILTER COEFFICIENT

C8040000	eng	ftype	coeff0	...	coeff15
----------	-----	-------	--------	-----	---------

The SET ALL REVERB FDN FILTER COEFFICIENT command sets the highpass / lowpass filter coefficients for all FDN lines on one reverb engine. The reverb engine number `eng` is 0 to 3 inclusive. The `ftype` parameter is 0 to instantiate a lowpass filter and 1 to instantiate a highpass filter. The command contains a list of 16 `coeff` coefficients for delay lines 0 to 15. The two's-complement 32-bit coefficient data is scaled so that `0x40000000 = +0.5`.

All multi-byte integers are [big-endian](#).

4.11.11 SET REVERB INPUT GAIN

B4040000	eng	chan	gain	ramp
----------	-----	------	------	------

The SET REVERB INPUT GAIN command sets the input gain and ramp time for one channel in one reverb engine. The reverb engine number `eng` is 0 to 3 inclusive. The channel number `chan` is 0 to 63 inclusive.

Amplitude values (`gain`) are expressed in units of 0.1 dB, with an offset of 1000. An amplitude value of 1000 is 0.0 dB; 900 is -10.0 dB. The maximum amplitude value is 1000, which is 0.0 dB. An amplitude value of zero is off ($-\infty$ dB).

Ramps are expressed in units of milliseconds. The input gain on the channel ramps from its current values to the value specified in the command over the specified time.

All multi-byte integers are [big-endian](#).

4.11.12 SET REVERB MATRIX GAIN

B5040000	eng	chan	gain	ramp
----------	-----	------	------	------

The SET REVERB MATRIX GAIN command sets the matrix gain and ramp time for one channel in one reverb engine. The reverb engine number `eng` is 0 to 3 inclusive. The channel number `chan` is 0 to 63 inclusive.

Amplitude values (`gain`) are expressed in units of 0.1 dB, with an offset of 1000. An amplitude value of 1000 is 0.0 dB; 900 is -10.0 dB. The maximum amplitude value is 1000, which is 0.0 dB. An amplitude value of zero is off ($-\infty$ dB).

Ramps are expressed in units of milliseconds. The input gain on the channel ramps from its current values to the value specified in the command over the specified time.

All multi-byte integers are [big-endian](#).

4.11.13 SET REVERB MIX FADER GAIN

C4040000	eng	gain
----------	-----	------

The SET REVERB MIX FADER GAIN command sets the mix fader gain on one reverb engine. The reverb engine `eng` is 0 to 3 inclusive.

Amplitude values (`gain`) are expressed in units of 0.1 dB, with an offset of 1000. An amplitude value of 1000 is 0.0 dB; 900 is -10.0 dB. The maximum amplitude value is 1000, which is 0.0 dB. An amplitude value of zero is off ($-\infty$ dB).

All multi-byte integers are [big-endian](#).

4.11.14 SET REVERB MATRIX FEEDBACK GAIN

BE040000	eng	delayLine	gain
----------	-----	-----------	------

The SET REVERB MATRIX FEEDBACK GAIN command sets the matrix feedback gain on one FDN delay line on one reverb engine. The reverb engine `eng` is 0 to 3 inclusive. The delay line number `delayLine` is 0 to 15 inclusive.

Amplitude values (`gain`) are expressed in units of 0.1 dB, with an offset of 1000. An amplitude value of 1000 is 0.0 dB; 900 is -10.0 dB. The maximum amplitude value is 1000, which is 0.0 dB. An amplitude value of zero is off ($-\infty$ dB).

All multi-byte integers are [big-endian](#).

4.11.15 SET ALL REVERB MATRIX FEEDBACK GAIN

C9040000	eng	gain0	...	gain15
----------	-----	-------	-----	--------

The SET ALL REVERB MATRIX FEEDBACK GAIN command sets the matrix feedback gain on all FDN delay lines on one reverb engine. The reverb engine `eng` is 0 to 3 inclusive. The command contains a list of 16 `gain` values for delay lines 0 to 15.

Amplitude values (`gain`) are expressed in units of 0.1 dB, with an offset of 1000. An amplitude value of 1000 is 0.0 dB; 900 is -10.0 dB. The maximum amplitude value is 1000, which is 0.0 dB. An amplitude value of zero is off ($-\infty$ dB).

All multi-byte integers are [big-endian](#).

4.11.16 SET REVERB OUTPUT GAIN

BF040000	eng	delayLine	chan	gain
----------	-----	-----------	------	------

The SET REVERB OUTPUT GAIN command sets the gain on one output channel, one FDN delay line, and one reverb engine. The reverb engine `eng` is 0 to 3 inclusive. The delay line number `delayLine` is 0 to 15 inclusive. Output channel `chan` is 0 to 63 inclusive.

Amplitude values (`gain`) are expressed in units of 0.1 dB, with an offset of 1000. An amplitude value of 1000 is 0.0 dB; 900 is -10.0 dB. The maximum amplitude value is 1000, which is 0.0 dB. An amplitude value of zero is off ($-\infty$ dB).

All multi-byte integers are [big-endian](#).

4.11.17 SET REVERB OUTPUT GAINS

C6040000	eng	delayLine	g0	...	g63
----------	-----	-----------	----	-----	-----

The SET REVERB OUTPUT GAINS command sets the gain on all 64 output channels, on one FDN delay line, and one reverb engine. The reverb engine `eng` is 0 to 3 inclusive. The delay line number `delayLine` is 0 to 15 inclusive.

Amplitude values (`gain`) are expressed in units of 0.1 dB, with an offset of 1000. An amplitude value of 1000 is 0.0 dB; 900 is -10.0 dB. The maximum amplitude value is 1000, which is 0.0 dB. An amplitude value of zero is off ($-\infty$ dB).

All multi-byte integers are [big-endian](#).

4.11.18 SET REVERB PRE FEEDBACK GAIN

C1040000	eng	gain
----------	-----	------

The SET REVERB PRE FEEDBACK GAIN command sets the pre-feedback gain on one reverb engine. The reverb engine `eng` is 0 to 3 inclusive.

Amplitude values (`gain`) are expressed in units of 0.1 dB, with an offset of 1000. An amplitude value of 1000 is 0.0 dB; 900 is -10.0 dB. The maximum amplitude value is 1000, which is 0.0 dB. An amplitude value of zero is off ($-\infty$ dB).

All multi-byte integers are [big-endian](#).

4.11.19 SET REVERB POST DELAY

BC040000	eng	delayLine	smpl
----------	-----	-----------	------

The SET REVERB POST DELAY command sets the delay on one post-reverb mix delay line.

The reverb engine `eng` is 0 to 3 inclusive. The delay line number `delayLine` is 0 to 15 inclusive. The `smpl` parameter is number of samples of delay divided by 8 (0 to 16383 inclusive).

Note that the current sample rate setting affects the translation between time and the `smpl` parameter. If the current sample rate in kHz is SR_{kHz} (either 48 or 96) and $delay_{ms}$ is the delay in milliseconds, the following equation applies:

$$smpl = \frac{delay_{ms} \times SR_{kHz}}{8}$$

All multi-byte integers are [big-endian](#).

4.11.20 SET REVERB PRE DELAY

BA040000	eng	smpl
----------	-----	------

The SET REVERB PRE DELAY command sets the pre-delay for one reverb engine

The `smpl` parameter is number of samples of delay divided by 8 (0 to 16383 inclusive).

Note that the current sample rate setting affects the translation between time and the `smpl` parameter. If the current sample rate in kHz is SR_{kHz} (either 48 or 96) and $delay_{ms}$ is the delay in milliseconds, the following equation applies:

$$smpl = \frac{delay_{ms} \times SR_{kHz}}{8}$$

All multi-byte integers are [big-endian](#).

4.11.21 SET REVERB PREFILTER COEFFICIENTS

B6040000	eng	band	ftype	gain	freq	bw
	B0	B1	A0	A1	A2	

The SET REVERB PREFILTER COEFFICIENTS command sets the coefficients for one band of EQ on one reverb engine. The reverb engine `eng` is 0 to 3 inclusive. There are eight filter bands available. `band` is 0 to 7 inclusive. Filter type `ftype` is defined by:

```
enum eFType { eFTypeBell = 0, eFTypeLowShelf, eFTypeHighShelf, eFTypeLowPass,
              eFTypeHighPass, eFTypeNotch, eFTypeBandpass};
```

The `gain` parameter is expressed as $dB \times 10$. For example, if the filter gain should be +14.6dB, the parameter is 146. The `freq` parameter is expressed as $f_c \times 100$. For example, if the filter centre frequency should be 225.75 Hz, the parameter is 22575. The `bw` parameter is expressed as $bw_{oct} \times 1000$. For example, if the filter bandwidth should be one octave, the parameter is 1000. The range for filter bandwidth is 300 to 3000 inclusive.

The coefficients `B0`, `B1`, `A0`, `A1`, `A2` are calculated based on the above parameters. See the section on [EQ](#).

All multi-byte integers are [big-endian](#).

4.11.22 SET REVERB PRE POST FLAGS

C3040000	eng	lowF	highF
----------	-----	------	-------

The SET REVERB PRE POST FLAGS command sets the pre / post flags for one reverb engine. There is a flag for each input channel fader / EQ instance. Lower flags `lowF` are for channels 0..31 and upper flags `highF` are for channels 32..63. If a flag bit is 0, the metering for the channel is set to pre-fader send; if the flag bit is 1, the metering for the channel is set to post-fader send.

All multi-byte integers are [big-endian](#).

4.11.23 SET REVERB PARAMETERS

CB040000	eng	n0	...	nn	mode	future
	min1	min2	min3	min4	min5	
	max1	max2	max3	max4	max5	

The SET REVERB PARAMETERS command sets parameters needed by the reverb GUI. The reverb GUI sends a SET REVERB PARAMETERS command to the unit whenever any underlying data changes. The unit firmware stores the data, and provides it as part of the data returned by [GET REVERB](#).

The reverb engine `eng` is 0 to 3 inclusive. The name field `n0 . . . nn` is exactly 20 characters in length. There must be a terminating null character; the field accommodates a string up to 19 characters in length. `mode` is a 32-bit integer, and is for use to set the reverb display mode. `future` is for future use. There are `min` (minimum) and `max` (maximum) values for each of five controls.

All multi-byte integers are [big-endian](#).

4.11.24 REVERB RECALL

A5040000	i1	p1	o1	i2	p2	o2	i3	p3	o3	i4	p4	o4	n0	...	nn	0
----------	----	----	----	----	----	----	----	----	----	----	----	----	----	-----	----	---

The REVERB RECALL command contains data needed by the reverb GUI when a cue containing a reverb preset is recalled. When a cue is generated in TiMax2 software, if the cue contains a reverb preset, one of these commands is put into the cue. When the cue is recalled in the TiMax2 unit, the command is executed. Executing this command causes a ReverbRecallNotification (see below) to be sent to all connected hosts. This notification message contains all the data in the command.

A 12-byte `include` boolean array follows the command longword. The boolean array contains input, preset and output booleans for each of the four reverb engines. These booleans indicate whether the input, preset and/or output data for each engine is included in the preset. The first boolean `i1` is the input boolean for engine one. The next boolean `p1` is the preset boolean for engine one. The last boolean `o4` is the output boolean for engine four.

A string containing the name of the cue follows the boolean array. This string is shown as `n0 . . . nn`. The null-terminated UTF-8 string is 60 characters or less in length, including the terminating null byte.

```
const int revEngines = 4;
const int revSnapshotCount = 3;
struct ReverbRecallNotification {
    int code; //ccReverbRecallNotification, 0x9C
    bool include[revEngines][revSnapshotCount];
    char cuename[60];
};
```

All multi-byte integers are [big-endian](#).

4.12 NOTIFICATIONS

The TiMax unit sends notification messages to all connected software hosts when certain events occur. Software monitoring incoming ethernet traffic should expect these notification messages. The most-significant byte of the incoming message identifies the notification.

AD	Show has been updated
7A	Cue has been triggered
9A	Cue clock was stopped
9B	Cue was stopped (clock may still be running)
7C	Disk error
94	MTC full in message
95	MTC start message
9E	Input location changed
9F	Panspace Point
7F	Sample Rate changed
B3	Show Transfer complete
7E	Temperature high

5 MIDI Commands

This section specifies TiMax2 MIDI commands.

The TiMax2 responds to a subset of standard MIDI commands (MSC and system exclusive strings are ignored). MIDI commands are received over the two ‘MIDI IN’ ports on the rear panel of the TiMax2. Some commands are only received on one or the other port, others are received over either port.

5.1 MIDI Messages

Except for system exclusive messages (which are not used by TiMax2), all MIDI messages are either two or three bytes in length. In a MIDI message, the first byte is called the status byte. The most-significant four bits of the status byte define the message type. The least-significant four bits of the status byte contain the MIDI channel number.

The status byte is followed by either one or two parameter bytes. Based on the message type, a receiving device knows whether to expect one byte after the status byte or two.

5.2 MIDI Channel

In the MIDI standard there are 16 channels defined. Within MIDI messages, these are numbered 0 through 15 inclusive. (This same set of numbers is sometimes presented to users as 1 to 16 inclusive. This specification is concerned only with numbers as contained in the messages.)

The MIDI channel number is placed in the least-significant four bits of the status byte.

Devices that receive MIDI messages, including TiMax2 units, can be set to a specific MIDI channel. When set to a specific MIDI channel, a MIDI receiving device such as the TiMax2 will ignore incoming MIDI messages to other channels. Devices that receive MIDI messages, including TiMax2 units, can also be set to receive on all MIDI channels. See the [SET MIDI CHANNEL](#) command.

5.3 MIDI Command Table

Table 1 specifies the three TiMax2 commands that can be controlled via MIDI messages. All numerical constants in the table are in hexadecimal. The ‘x’ in the Status byte column refers to the MIDI channel.

1. Group levels can be set via MIDI Continuous Controller messages received on TiMax2 MIDI port 1. In MIDI Continuous Controller messages, the status byte is 0xB0 through 0xBF (MIDI channels 0 through 15). TiMax2 interprets the second byte as the group number and the third byte as the amplitude. The standard MIDI amplitude range, 0..0x7F, is mapped onto the native TiMax amplitude range 0..1100. See [SET GROUP LEVEL](#).
2. Image definitions can be recalled via MIDI continuous controller messages received on TiMax2 MIDI port 2. See [RECALL IMAGE DEFINITION](#). The global tracking enable must be set: [SET TRIGGER ENABLES](#). TiMax2 interprets the second byte as the input channel number and the third byte as the image definition tracking number. Ramps are set by [SET TRACKING RECALL RAMP](#).
3. Cues can be triggered via MIDI Note On messages. In MIDI Note On messages, the status byte is 0x90 through 0x9F (MIDI channels 0 through 15). The cue to be triggered has been programmed in the show file to respond to MIDI Note On messages with the value specified in the second byte (see ‘Programming a MIDI Trigger’, following page). The third byte must be any number between 0x40 and 0x7F.

Cues can also be triggered via MIDI Program Change messages. In MIDI Program Change messages, the status byte is 0xC0 through 0xCF (MIDI channels 0 through 15). The cue to be triggered has been programmed in the show file to respond to MIDI Program Change messages the value specified in the second byte.

TiMax2 Command	Port	MIDI Message	Status byte	2nd byte	3rd byte
Set Group Level	1	Continuous Controller	Bx	Group 0..1F	Amplitude 0..7F
Recall Image Def	2	Continuous Controller	Bx	Channel 0..3F	Tracking # 0..7F
Trigger Cue	either	Note On	9x	0..7F	Amplitude > 40
Trigger Cue	either	Program Change	Cx	0..7F	none

Table 1: MIDI Commands

5.4 Programming a MIDI Trigger

Figure 3 shows the TiMax2 Cue Editor being used to program Cue 12.4 to be triggered by a MIDI Program Change message on MIDI channel 3, with the second byte containing a 15 (0x0F).

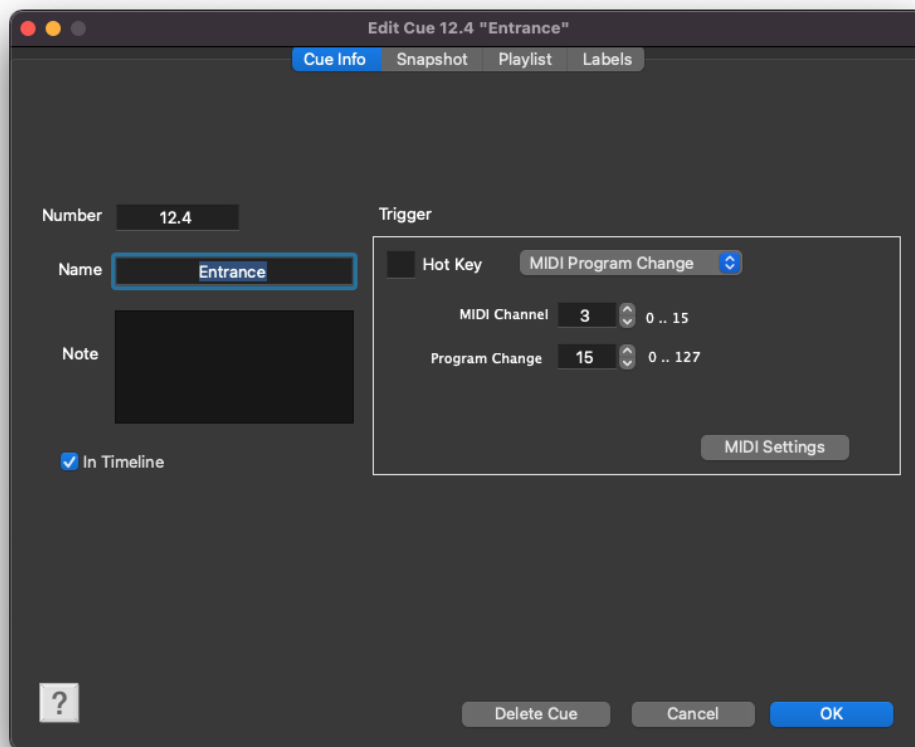


Figure 3: Programming a MIDI trigger in the TiMax2 Cue Editor

6 File Transfer

A set of routines is provided for software programs on various platforms to transfer files to and from the TiMax2 hardware. The TiMax2 software repository contains a directory `FileTransfer`, which contains the code for these routines. This code is written in plain 'C' language for portability. The only library dependency, other than the standard 'C' library, is for `zlib`, the file compression library. `zlib` is normally available to 'C' language linkers (e.g. in `/usr/local/lib` or equivalent).

The file `fileTransfer.c` contains the essential code. The header file `fileTransfer.h` defines the interfaces. The `FileTransfer` directory also contains 'make' files, a 'main.c' file for testing, test files and test scripts.

Note for compiling on Windows:

There is a line near the beginning of `fileTransfer.c`:

```
#define MACOS true
which must be replaced with
#define MACOS false
```

6.1 File Transfer Routines

The code is provided in `fileTransfer.c`, with the call interfaces (function prototypes) for the externally-visible routines specified in `fileTransfer.h`. These are:

6.1.1 Initialization: `initializeFileTransfer()`

Before transferring files, the software must call `initializeFileTransfer()`, passing a string containing the IP address of the TiMax2 unit (e.g. "192.168.1.22") This routine opens a socket connection to the TiMax2 unit, and initializes data structures.

6.1.2 Sending Files: `transmitFile()`

To send a file to the TiMax2 unit, the software calls `transmitFile()`, passing a string containing the full path name of the file to be transferred. Path names must use the forward slash ('/') as the directory separator on MacOS and the back slash ('\') as the directory separator on Windows. The name of the file must include a file extension that defines the file type. File names (after removing the extension) must be unique – when a file already exists on the TiMax2 disk, sending the file to the unit overwrites the previous version of the file.

The function is synchronous. When the function returns, the file has been written to the TiMax2 disk, unless an error occurred. A return value from the function encodes the status of the operation. There is also a function, `errorText()`, that provides strings for each of the error codes.

6.1.3 Receiving Files: `receiveFile()`

To receive a file from the TiMax2 unit, the software calls `receiveFile()`, passing a string containing the full path name of the file to be written to the user's disk. The name of the file must include a file extension that defines the file type. This file will be created if it does not already exist. The file name portion of the path, including extension, must match the name of the file when it was transmitted.

The function is synchronous. When the function returns, the file has been received from the TiMax2 unit and written to the local file specified, unless an error occurred. A return value from the function encodes the status of the operation. There is also a function, `errorText()`, that provides strings for each of the error codes.

Panspace background graphic files are associated with the `show` file that uses them. For these files, in addition to the full path name of the graphic file to be written to the user's disk, the name of the `show` is passed to the routine in the `sfn` parameter. For all other file types, a zero is passed as the `sfn` parameter.

6.2 File Types

A number of file types are supported. When files are specified to `transmitFile()` or `receiveFile()`, the file names must use one of the extensions shown in Table 2, which set the file's type in firmware.

Table 2 indicates which file types are supported by the `FileTransfer` routines for sending and receiving.

Binary show files are written by TiMax2 software only (not supported by the `FileTransfer` routines, as shown in Table 2).

File Type	Extension	Send	Receive
Audio for playback	.tma1, .tma2, etc.	Y	Y
Show (binary with XML embedded)	.bsf	N	N
Firmware	.TMX	Y	N
Audio Waveform	.wfm	Y	Y
System Error Log	.serr	N	Y
XML for show open on TiMax	.tm2	N	Y
Panspace Background Graphic	.jpg, .jpeg, .png, .svg, .svgz, .tif, .tiff	Y	Y
FPGA Configuration	.bit	Y	N
Firmware Memory Dump	.fwm	N	Y
Reverb GUI Settings	.rev	Y	Y

Table 2: File Types

Note 1: 'External' audio file formats '.wav', '.aif' and '.aiff' are converted to the TiMax2 native audio format '.tma n' prior to calling `transmitFile()`. The number of audio channels in the file is included in the extension: '.tma1' for mono, '.tma2' for stereo, etc. In TiMax2 software, routines in `FileConvert.cpp` performs these conversions.

Note 2: Reverb GUI Settings files, using extension '.rev', are compressed before being transmitted, and are uncompressed after having been received and before being written to local disk.