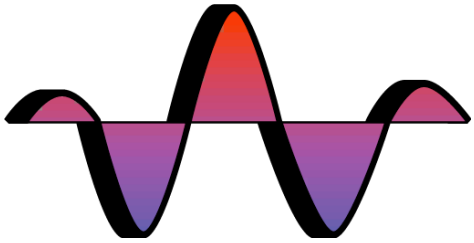

TIMAX2

COMMAND PROGRAMMING

Software versions 3.9.x, 4.x.x, and 5.x.x March, 2019

OutBoard Electronics Unit4 Church Meadows, Haslingfield Road, Barrington, Cambs CB22 7RG, UK

Ethernet Address Protocols

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

When using default settings (i.e. no fixed-IP set), 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 (often hosted within the network's router). If no response is received from a DHCP server, the unit will self-assign an unused IP address after a short delay. These methods enable the unit to acquire a unique 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 the unit's front panel (or from the Information / Configuration window Hardware tab in software). When a fixed IP has been set, the unit does not use DHCP and does not self-assign an address during boot up.

When launched, TiMax software discovers TiMax units on its local area network by broadcasting an "inquiry" packet to the TiMax port (0xE7C3). Generally, a TiMax unit and the computer running the TiMax software must be on the same subnet for the packet to reach the TiMax unit. (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.) 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. 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.

Ethernet Commands

This section specifies transmission and formatting details for commands commonly used to control the TiMax2 SoundHub via the ethernet port. These commands are a small subset of the total TiMax2 command set, but include all commands needed to build interfaces that control levels, including mutes and solos, and that run cues.

The command formats specified in the table below comprise the data portion of a UDP packet. One TiMax2 command is sent per UDP packet. Only UDP packets with valid IP address, port, CRC, IP checksum and UDP checksum are processed by the TiMax2 firmware. (CRC generation is normally done in ethernet hardware. IP and UDP checksums are normally generated in the UDP socket software. Packet programmers do not usually have to be concerned with these details.)

The first two bytes of command data are always 0x7D 0x00 (except the Inquiry command). This two-byte prefix is used to 32-bit align all following data. The third and fourth bytes specify the TiMax2 command code. Command parameters occupy the bytes following the command code. Multi-byte binary numbers are sent in big-endian format, as described in the following tables.

Ethernet Command Format Table All constant values are displayed in hexadecimal.

Command	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Group Level	7D	00	44	04	00	00	00	00	00	*gn*	00	00	*am*	*al*
Group Solo	7D	00	5B	04	00	00	00	00	00	*gn*	00	00	00	0/1
Group Mute	7D	00	5A	04	00	00	00	00	00	*gn*	00	00	00	0/1
Input Level	7D	00	41	03	00	*ch*	00	00	*rm*	*rl*	00	00	*am*	*al*
Input Solo	7D	00	47	04	00	*ch*	00	00	00	0/1				
Input Mute	7D	00	45	04	00	*ch*	00	00	00	0/1				
Output Level	7D	00	42	04	00	*ch*	00	00	*rm*	*rl*	00	00	*am*	*al*
Output Solo	7D	00	48	04	00	*ch*	00	00	00	0/1				
Output Mute	7D	00	46	04	00	*ch*	00	00	00	0/1				
Mute All Outputs	7D	00	43	04	00	00	00	00	00	0/1				
Stop Cues	7D	00	28	05	00	00								
Go Cue	7D	00	03	04	00	00	*nm*	*nl*	*d1*	*d2*	00	00	00	00
Get IO Levels	7D	00	68	06	00	00								
Get Solo Mute	7D	00	75	06	00	00								
Get Status	7D	00	6F	06	00	00								
Auto-update	7D	00	85	06	00	00	00	00	00	01	00	00	00	00
Inquiry	6E	A7	00	04										
Pause Playback	7D	00	23	04	00	00	00	00	00	*clo*	00	00	00	*chi*
Resume Playback	7D	00	22	04	00	00	00	00	00	*clo*	00	00	00	*chi*
Set Realtime Clock	7D	00	83	06	00	00	00	*yr*	*mh*	*dy*	*dow*	*hr*	*mn*	*sc*
Recall Image Def	7D	00	3E	04	00	00	*ch*	*nm*	*nl*					
Panspace Point	7D	00	5F	02	00	*ch*	00	00	*rm*	*rl*	00	00	*xm*	*xl*
(continued)	00	00	*ym*	*yl*	00	00	*nm*	*nl*	00	00	*ssm*	*ssl*		

gn	Group numbers are 0..31 (0x00..0x1F). Group number zero is normally labeled “1” to the user, etc.
am, al	Amplitude is expressed in units of 0.1 dB, with an offset of 1000. Thus an amplitude value of 1000 is 0.0 dB. The maximum amplitude value is 1100, which is +10.0 dB. An amplitude value of 900 is -10.0 dB. An amplitude value of zero is off (minus infinity). The amplitude value occupies two bytes in the command, with the most significant byte first. For example, the amplitude value for 0.0 dB is 1000, which in hex is 0x3E8, *am* would be 0x03 and *al* would be 0xE8.
rm, rl	Ramps are expressed in units of milliseconds. The ramp value occupies two bytes in the command, with the most significant byte first. For example, a one second ramp is 1000 milliseconds, which in hex is 0x3E8, *rm* would be 0x03 and *rl* would be 0xE8.
ch	Channel numbers are 0..63 (0x00..0x3F). Channel number zero is normally labeled “1” to the user, etc.
0/1	Solos and mutes are activated with a “1” and deactivated with a “0”.

nm, nl, d1, d2	Valid cue numbers are in the range 1..65531. The cue number occupies two bytes in the command, with the most significant byte first. For example, a cue number of 858, which in hex is 0x35A, *nm* would be 0x03 and *nl* would be 0x5A. A cue number can have one or two decimal sub numbers, e.g. “858.3.12”. If these are unused, *d1* and *d2* are set to zero. If d2 is in use (non-zero), then d1 is also in use. Sub numbers can go up to 255. The entire four-byte number for cue 858.3.12, as an example, would be 0x03 0x5A 0x03 0x0C.
Auto-update	An auto-update message requests the unit to send an update message whenever any externally-visible data changes. In addition to the status reply formats specified below, update messages sent via auto-update include delay settings, playback status, module settings, etc. When using auto-update, it is necessary to filter the desired data out from all of the data that is sent. If an auto-update message is sent once every second or so, update messages will continue to be sent. If no auto-update message is received after approximately 5 seconds, auto-update times out and no further messages are sent until another auto-update message is received.
Inquiry	Broadcast an Inquiry message to discover all TiMax units on a local network. An Inquiry reply will be sent by every TiMax that is reachable on the network (i.e. on the same subnet).
clo, chi	Pause Playback and Resume Playback operate on a range of playback channels. The first channel number (clo) is the lowest channel of the range, and the second channel number (chi) is the highest channel of the range. Channel numbers are 0..63 (0x00..0x3F). Channel number zero is normally labeled “1” to the user, etc. Pause playback on all channels, for example, is 0x7D 0x00 0x23 0x04 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x3F. Resume Playback has an effect on a channel only if that channel has an audio file loaded (for example, a previously-triggered cue loaded and started playback on a channel, and a Pause Playback command paused playback on that channel).
Setting Real-time clock	All values are encoded in BCD: This is a C function for BCD: char bcd (int n) { return (((n/10) << 4) + (n % 10)); } *yr*: year - 2000 e.g. 16 for the Gregorian calendar year 2016 (breaks in 2100, but byte before yr is in reserve) *mh*: month: 1: January – 12: December *dy*: day of month: 1 – 31 *dow*: day of week: 1: Monday – 7: Sunday *hr*: hour: 0 – 23 *mn*: minute: 0 – 59 *sc*: second: 0 – 59
Recall image	The image definition number occupies two bytes in the command, with the most significant byte first. For example, to recall image definition 300, which in hex is 0x12C, nm would be 0x01 and nl would be 0x2C. The ramp time used is set in the system preset (Tracking Recall Image Definition Ramps).

xm, xl, ym, yl, nm, nl, ssm, ssl	Panspace point command parameters are: channel, ramp, x coordinate, y coordinate, inclusion range and subspace number. Panspace point coordinates are expressed in values from 0 to 2896. The firmware calculates an interpolation based on all the distances from the panspace point to all the image definitions in the panspace. Other than channel, which is a single byte, each parameter occupies two bytes in the command, with the most significant byte first. For example, an X coordinate of 1126, which in hex is 0x466, xm would be 0x04 and xl would be 0x66. Only image definitions within the inclusion range of the point are included in the interpolation (except that the two closest image definitions are always included). The inclusion range parameter is sent as an inverse inclusion distance. The inclusion range is displayed in the user interface as a number between 1 and 100. To generate the parameter (nm, nl) for the Panspace Point command, this range number is multiplied by the maximum coordinate value 2896 then divided by 100. The result is the radius of the inclusion range circle in panspace coordinates. Then the maximum value of an unsigned 16-bit number, 0xFFFF, is divided by the inclusion circle radius to get the inverse distance. As an example, for an inclusion range number of 5, the radius is 145. 0xFFFF / 145 is 452, which in hex is 0x1C4, so nm would be 0x01 and nl would be 0xC4. The subspace number 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. In this case the subspace number should be zero. If subspaces have been defined, then the show file XML should be opened and the subspace list located (search on <subspaces>). With one subspace defined, it looks something like this: <subspaces>Space 1,All</subspaces>. The list is in the order of subspace number, starting at zero. In this case, “Space 1” is subspace zero and “All” is subspace 1.

Ethernet Status Reply Formats All constant values are hexadecimal.

Reply	1	2	3	4	5	6	7					size
Get IO Levels	7D	00	68	06	00	00	*am*[1]	*al*[1]	…	…	…	6+896 +14
Get Solo Mute	7D	00	75	06	00	00	*ms*[2]	*ms*[2]	…	…	…	6+192
Inquiry	7D	00	6E	A7	00	04	IP[3]	IP[3]	…	…	…	
Get Status	7D	00	6F	06	00	00	*fr*[4]	*fr*[4]	…	…	…	6+256

1. Amplitude is expressed in units of 0.1 dB, with an offset of 1000. Thus an amplitude value of 1000 is 0.0 dB. The maximum amplitude value is 1100, which is +10.0 dB. An amplitude value of 900 is -10.0 dB. An amplitude value of zero is off (minus infinity). The amplitude value occupies two bytes in the response, with the most significant byte first. For example, the amplitude value for 0.0 dB is 1000, which in hex is 0x3E8, *am* would be 0x03 and *al* would be 0xE8.

   The structure of the Get IO Levels data, beginning at byte 7, is expressed in C language syntax as follows (note all short and int values are in big-endian byte-order; short is 16-bit and int is 32-bit)

   ```
   struct {
     short analogSourceLevel[64]; //(0..1000)
     short moduleSourceLevel[64]; //(0..1000)
     short playbackSourceLevel[64]; //(0..1000)
     short inputGainLevel[64]; //(0..1100)
     short outputGainLevel[64]; //(0..1100)
     short imageDefinitionNumber[64]; //(0..512)
     int groupLevel[32]; //(0..1100)
     char unused[14];
     };
   ```

   Note that the group levels are expressed as 4-byte integers, in big-endian byte-order.

2. The structure of the solo /mute data has not changed from the original TiMax2 firmware.
   Input and output channel mute / solo data is returned as one byte of data per channel. Each byte contains a code in the range 0..7. Each of the three bits in use encodes mute / solo state information as follows: (1 is on and 0 is off)
   bit 0: mute
   bit 1: solo
   bit 2: solo-muted. A channel is solo-muted when any other input / output channel is soloed.
   Group channel mute / solo data is returned as one byte of data per channel. Each byte contains either a zero or a one (1 is on and 0 is off).

   The structure of the Get Solo Mute data, beginning at byte 7, is expressed in C language syntax as follows:

   ```
   struct {
     char inputChannels[64];
     char outputChannels[64];
     char groupMute[32];
     char groupSolo[32];
     };
   ```

3. The structure of the returned Inquiry data, beginning at byte 7, is expressed in C language syntax as follows: (note all short and int values are in big-endian byte-order; short is 16-bit and int is 32-bit)

```
struct {
  int IPaddress;
  short inputChannels;
  short outputChannels;
  char designName[24];
  char firmwareVersion[4];
  char firmwareDateTime[16];
  char serialNumber[8];
  char unitName[32];
  …
  };
```

4. The structure of the returned Status data, beginning at byte 7, is expressed in C language syntax as follows: (note all short and int values are in big-endian byte-order; short is 16-bit and int is 32-bit)

```
struct {
  int freeIOQueueEntries;  //count of free IO queue entries
  int freeIOQueueLWM;  //low water mark - free IO queue entries
  int timerQueueEntries;  //number of entries on the timer queue
  int workQueueEntries;  //number of entries on the work queue
  int midi1QueueEntries;  //number of entries on the MIDI 1 TX queue
  int midi2QueueEntries;  //number of entries on the MIDI 2 TX queue
  int currentTime;  //current clock (millisecond count from power on)
  int showClock; //show clock
  int showMemoryUsage;  //show memory usage
  int cpuUtilization;  //cpu utilization in percent
  int enetCmdMsgs;  //ENET cmd string messages received
  int enetCmdMsgsOK;  //ENET good cmd string messages received
  int enetCmdMsgsBad;  //ENET invalid cmd string messages received
  int midi1RxMsgs;  //MIDI port 1 messages received and passed to handler
  int midi1RxMsgsOK;  //MIDI port 1 good messages received (unused)
  int midi1RxMsgsBad;  //MIDI port 1 bad messages received
  int midi2RxMsgs;  //MIDI port 2 messages received and passed to handler
  int midi2RxMsgsOK;  //MIDI port 2 good messages received (unused)
  int midi2RxMsgsBad;  //MIDI port 2 bad messages received
  int midi1TxMsgs;  //MIDI port 1 messages transmitted
  int midi2TxMsgs;  //MIDI port 2 messages transmitted
  int unitTemperature;  //unit temperature, degrees C
  int taskNoBufferErrors; //task no buffer errors
  int showQueueEntries;  //number of entries on the show queue
  int activePlaybackChannels;  //number of active playback channels
  int stalledPlaybackBlocks;  //blocks not sent waiting for HDD
  char cueClockRunning;  //cue clock running
  char showClockRunning;  //show clock running
  char graphicInShow;  //show contains a graphic file
  char unused;
  int diskRequestQueueCount;  //count of entries in disk request queue
  int freeDiskRequestEntries;  //number of free DRPBs
  int freeDiskRequestQueueLWM;  //number of free DRPBs low water mark
  int badDiskRequestCount;  //count of bad disk manager requests
  int GPI_Pins;  //current state of GPI port pins
  int bootDelay;  //programmed boot up delay, seconds
  int diskReads;  //disk read count
  int diskWrites;  //disk write count
  int receivedPackets;  //received ethernet packet count
  int transmittedPackets;  //transmitted ethernet packet count
  int ethernetNoBufferErrors;  //ethernet receive no buffer errors
  int ethernetRxOverrunErrors;  //ethernet receive overruns
  int ethernetTxBufferAllocations;  //ethernet transmit buffer allocations
  int ethernetRxCRCErrors;  //ethernet receive CRC errors
  int ethernetRxChecksumErrors;  //ethernet checksum errors
  int zeroConfigAttempts;   //zero configuration attempts
  int ethernetRxNonOctetErrors;  //ethernet receive non-octet errors
  int currentBackupFileNumber;  //backup number (0: no backups pending)
  int ethernetRxTooLargeErrors;  //ethernet receive packet too large errors
  int cueClock;  //cue sequence clock
  int ethernetRxTruncatedErrors;  //ethernet receive truncated receive errors
  int showFileFormatErrors;  //count of show file format errors
  int numberOfLiveCommands;  //entries in the live commands list for show
  char realTimeString[23];  //real time display string
  char batteryLow;  //real time clock battery low
  char incomingMTCrunning;  //incoming MTC running and valid
  char incomingMTCframeRate;  //incoming MTC frame rate bits (0..3)
  char incomingMTChours;  //incoming MTC hours
  char incomingMTCminutes;  //incoming MTC minutes
  char incomingMTCseconds;   //incoming MTC seconds
```

```
  char incomingMTCdframes;  //incoming MTC frames
  char MTCGeneratorRunning;  //MTC generator running
  char MTCGeneratorFrameRate;  //MTC generator frame rate type (0..3)
  char MTCGeneratorHours;  //MTC generator hours
  char MTCGeneratorMinutes;  //MTC generator minutes
  char MTCGeneratorSeconds;  //MTC generator seconds
  char MTCGeneratorFrames;  //MTC generator frames
  int currentShow;  //file number of currently loaded show (0 means none)
  int currentCueNumber;  //current cue number (last cue to be triggered)
  int numberOfLiveCues;  //entries in the live cues list
  int DHCPLeaseTime;  //DHCP lease time
  int incomingMTCtime;  //incoming MTC in milliseconds
};
```

(For cue number format, see *Go Cue*, above, bytes 7 through 10.)

MIDI Commands

This section specifies TiMax2 SoundHub MIDI commands. The TiMax2 responds to a subset of standard MIDI commands (MSC and sysex 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.

MIDI Command Format Table All constant values in hexadecimal

Command	Port	MIDI Message Type	2nd byte	3rd byte
Set Group Level	1	Controller (Bx)[1]	Group[2] 0..1F	Amplitude[3] 0..7F
Image Definition Recall	2	Controller (Bx)[1]	Channel[4] 0..63	Image Definition 0..7F
Trigger Cue	either	Note On (9n)[5]	trigger setting	Amplitude > 40
Trigger Cue	either	Program Change (Cn)[5]	trigger setting	none

1	The MIDI channel number is ignored. The range 0xB0..0xBF is valid as the first byte for controller message.
2	Group numbers are 0..31 (0x00..0x1F). Group number zero is normally labeled "1" to the user, etc.
3	Amplitude is expressed in the standard MIDI range 0..7F. This is mapped onto the native TiMax amplitude range 0..1100.
4	TiMax channel numbers are 0..63 (0x00..0x3F). Channel number zero is normally labeled "1" to the user, etc.
5	Cues are triggered using either Note On or Program Change messages. The MIDI message to trigger a particular cue is set in the trigger section of the cue editor in the TiMax2 software by specifying (1) either Note On or Program Change, (2) the MIDI channel number and (3) the value in the second byte of the MIDI message. The MIDI channel number (0x00..0x0F) is the lower four bits of the first byte of the MIDI message ('n' in the table). A Program Change message is two bytes in length. A Note On message has a third byte which can be any value greater than 64 (0x40).