Thorlabs APT Controllers

Host-Controller Communications Protocol

Date: 01-June-2012

Contents

Introduction

1. Purpose and Scope

This document describes the low-level communications protocol and commands used between the host PC and controller units within the APT family. The information contained in this document is intended to help third party system developers to write their own applications to interface to the Thorlabs range of controllers without the constraints of using a particular operating system or hardware platform. The commands described here are those which are necessary to control movement; there is an additional set of commands, used for calibration or test, which will not be detailed as these are not required for the external system developer.

2. Electrical interface

The APT family of controllers provides a USB and an RS-232 interface to communicate with the host PC. The communications protocol is identical in both cases but developers wishing to use the USB interface should be aware of the USB enumeration scheme used in the system.

2.1 USB Interface

The electrical interface within the APT controllers uses a Future Technology Devices International (FTDI), type FT232BM USB peripheral chip to communicate with the host PC. This is a USB2.0 compliant USB1.1 device. This USB interfacing chip provides a serial port interface to the embedded system (i.e. APT controller) and USB interface to the host control PC. While the overall communications protocol is independent of the transport layer (for example, Ethernet or serial communications could also be used to carry commands from the host to the controller), the initial enumeration scheme described below is specific to the USB environment.

FTDI supply device drivers and interfacing libraries (for Windows, Linux and other platforms) used to access the USB chip. Before any PC USB communication can be established with an APT controller, the client program is required to set up the necessary FTDI chip serial port settings used to communicate to the APT controller embedded system. Within the APT software itself the following FTDI library calls are made to set up the USB chip serial port for each APT USB device enumerated on the bus:-

```
// Set baud rate to 115200.
ftStatus = FT_SetBaudRate(m_hFTDevice, (ULONG)uBaudRate);

// 8 data bits, 1 stop bit, no parity
ftStatus = FT_SetDataCharacteristics(m_hFTDevice, FT_BITS_8, FT_STOP_BITS_1,
FT_PARITY_NONE);

// Pre purge dwell 50ms.
Sleep(uPrePurgeDwell);

// Purge the device.
ftStatus = FT_Purge(m_hFTDevice, FT_PURGE_RX | FT_PURGE_TX);

// Post purge dwell 50ms.
Sleep(uPostPurgeDwell);
```

```
// Reset device.
ftStatus = FT_ResetDevice(m_hFTDevice);

// Set flow control to RTS/CTS.
ftStatus = FT_SetFlowControl(m_hFTDevice, FT_FLOW_RTS_CTS, 0, 0);

// Set RTS.
ftStatus = FT_SetRts(m_hFTDevice);
```

2.2 USB Device Enumeration

The APT Server PC software supplied is designed to work with a number of different types of controller. The purpose of the enumeration phase is for the host to establish what devices are present in the system and initialise the GUI accordingly. Initially this is done by enumerating the USB devices connected to the system and reading the serial number information contained in the USB device descriptor.

For the Thorlabs range of controllers, this serial number is an 8-digit decimal number. The first two digits (referred to as the prefix) describe the type of controller, while the rest of the digits make up a unique serial number. By extracting the prefix, the host can therefore establish what type of hardware is connected to the system.

In most cases, specifically with benchtop controllers, the USB serial number contains sufficient information for the host to know the exact type of hardware is connected. There is a range of other controller products where several controller cards (without their own individual USB peripheral chip) can be plugged into a motherboard and it is only the motherboard that has USB connectivity. These are generally referred to as a card slot (or bay) type of system (for example, the BSC103 controller). In these systems, a second enumeration state is carried out; however, this second state is done within the protocol framework that will be detailed in this document.

For the controller types, the USB prefixes can be the following:

USB S/N	Type of product	Thorlabs code
20xxxxxx	Legacy single channel stepper driver	BSC001
25xxxxxx	Legacy single channel mini stepper driver	BMS001
30xxxxxx	Legacy dual channel stepper driver	BSC002
35xxxxxx	Legacy dual channel mini stepper driver	BMS002
40xxxxxx	Single channel stepper driver	BSC101
60xxxxxx	OptoSTDriver (mini stepper driver)	OST001
63xxxxxx	OptoDCDriver (mini DC servo driver)	ODC001
70xxxxxx	Three channel card slot stepper driver	BSC103
80xxxxxx	Stepper Driver T-Cube	TST001
83xxxxxx	DC servo driver T-Cube	TDC001
73xxxxxx	Brushless DC motherboard	BBD102/BBD103
94xxxxxx	Brushless DC motor card	BBD102/BBD103

Of these listed above, currently only the BSC103 (serial number prefix 70) and the BBD10x are card slot type of controllers.

2.3 RS-232 Interface

The RS-232 interface uses the 9-way D-Type male connector on the rear panel, marked 'INTERCONNECT'. Communications parameters are fixed at:

- 115200 bits/sec
- 8 data bits, 1 stop bit
- No parity
- No handshake

By nature, the RS-232 interface provides point-to-point communications, and therefore there is no device enumeration as there is with USB based communications.

3. Overview of the Communications Protocol

The communications protocol used in the Thorlabs controllers is based on the message structure that always starts with a fixed length, 6-byte *message header* which, in some cases, is followed by a variable length *data packet*. For simple commands, the 6-byte message header is sufficient to convey the entire command. For more complex commands, for example, when a set of parameters needs to be passed on, the 6 byte header is not enough and in this case the header is followed by the data packet.

The header part of the message always contains information that indicates whether or not a data packet follows the header and if so, the number of bytes that the data packet contains. In this way the receiving process is able to keep tracks of the beginning and the end of messages.

Note that in the section below describing the various byte sequences, the C-type of notation will be used for hexadecimal values (e.g. 0x55 means 55 hexadecimal) and logical operators (e.g. | means logic bitwise OR). Values that are longer than a byte follow the Intel little-endian format.

4. Description of the message header

The 6 bytes in the message header are shown below:

Byte:	byte 0	byte 1	byte 2	byte 3	byte 4	byte 5
Meaning if no data packet to follow	message ID		param1	param2	dest	source
Meaning if data packet to follow	message ID		data packet length		dest | 0x80	source

The meaning of some of the fields depends on whether or not the message is followed by a data packet. This is indicated by the most significant bit in byte 4, called the destination byte*,* therefore the receiving process must first check if the MSB of byte 4 is set.

If this bit is not set, then the message is a header-only message and the interpretation of the bytes is as follows:

message ID: describes what the action the message requests
param1: first parameter (if the command requires a parameter, otherwise 0)

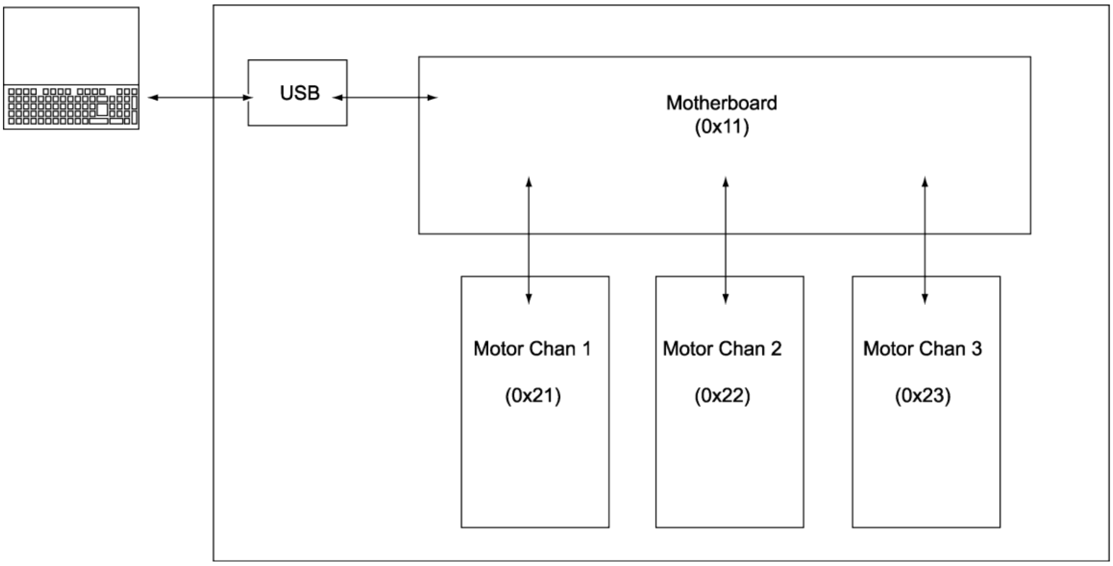
param2: second parameter (if the command requires a parameter, otherwise 0)
dest: the destination module
source: the source of the message

The meaning of the source and destination bytes will be detailed later.
If the MSB of byte 4 is set, then the message will be followed by a data packet and the interpretation of the header is the following:

message ID: describes what the action the message requests
datapacket length: number of bytes to follow after header
Note: although this is a 2-byte long field, currently no datapacket exceeds 255 bytes in length.
dest: | 0x80 the destination module logic OR'd with 0x80 (noted by d|)
source: the source of the data

The source and destination fields require some further explanation. In general, as the name suggests, they are used to indicate the source and destination of the message. In non-card-slot type of systems the source and destination of messages is always unambiguous, as each module appears as a separate USB node in the system. In these systems, when the host sends a message to the module, it uses the source identification byte of 0x01 (meaning host) and the destination byte of 0x50 (meaning "generic USB unit"). (In messages that the module sends back to the host, the content of the source and destination bytes is swapped.)

In card-slot (bay) type of systems, there is only one USB node for a number of sub-modules, so this simple scheme cannot be used. Instead, the host sends a message to the motherboard that the sub-modules are plugged into, with the destination field of each message indicating which *slot* the message must be routed to. Likewise, when the host receives a message from a particular sub-module, it knows from the source byte which slot is the origin of the message – see Fig below.



Numerically, the following values are currently used for the source and destination bytes:

0x01	Host controller (i.e control PC)
0x11	Rack controller, motherboard in a card slot system or comms router board
0x21	Bay 0 in a card slot system
0x22	Bay 1 in a card slot system
0x23	etc.
0x24	etc.
0x25	etc.
0x26	etc.
…	
0x2A	Bay 9 in a card slot system
0x50	Generic USB hardware unit

In slot-type systems the host can also send messages to the motherboard that the sub-modules are plugged into (destination byte = 0x11). In fact, as a very first step in the communications process, the host must send a message to the motherboard to find out which slots are used in the system.

Note that although in theory this scheme would allow communication between individual sub-modules (the source of the message could be a sub-module and the destination another one), current systems do not use this option.

5. General message exchange rules

The type of messages used in the communications exchange between the host and the sub-modules can be divided into 4 general categories:

(a) Host issues a command, sub-module carries out the command without acknowledgement (i.e. no response is sent back to the host).

Typically, these are commands which require no information from the sub-module, for example setting the digital outputs to a particular state.

(b) Host issues a command (message request) and the sub-module responds by sending data back to the host.

For example, the host may request the sub-module to report the state of the digital inputs.

(c) Following a command from the host, the sub-module periodically sends a message to the host without further prompting.

These messages are referred to as *status update messages*. These are typically sent automatically every 100 msec from the sub-module to the host, showing, amongst other things, the position of the stage the controller is connected to. The meters on the APT User GUI rely on these messages to show the up-to-date status of the stage.

(d) Rarely – error messages, exceptions. These are spontaneously issued by the sub-module if some error occurs. For example, if the power supply fails in the sub-module, a message is sent to the host PC to inform the user.

Apart from the last two categories (status update messages and error messages), in general the message exchanges follow the SET -> REQUEST -> GET pattern, i.e. for most commands a trio of messages are defined. The SET part of the trio is used by the host (or, sometimes in card-slot systems the motherboard) to set some parameter or other. If then the host requires some information from the sub-module, then it may send a REQUEST for this information, and the sub-module responds with the GET part of the command. Obviously, there are cases when this general scheme does not apply and some part of this message trio is not defined. For consistency, in the description of the messages this SET->REQUEST->GET scheme will be used throughout.

Note that, as the scheme suggests, this is a master-slave type of system, so sub-modules never send SET and REQUEST messages to the host and GET messages are always sent to the host as a destination.

In all messages, where a parameter is longer than a single character, the bytes are encoded in the Intel format, least significant byte first.

6. Format Specifiers

format	encoding
long	4 bytes in the Intel (big-endian) format for example decimal 123456789 (75BCD15H) is encoded as the byte sequence 15H, CDH, 5BH, 07H
short	2 bytes (4 digits) in the Intel (big-endian) format for example decimal 12345 (3039H) is encoded as the byte sequence 39H, 30H
word	2 bytes (4 digits) in the Intel (big-endian) format for example decimal 12345 (3039H) is encoded as the byte sequence 39H, 30H
dword	4 bytes in the Intel (big-endian) format for example decimal 123456789 (75BCD15H) is encoded as the byte sequence 15H, CDH, 5BH, 07H
char	1 byte (2 digits)
char[N]	string of N characters

7. Command Reference

In general, the messages used in the communication protocol can be divided into 5 main groups: generic commands, move parameter setup commands, move initiating commands, status update message related commands and error messages.

The commands listed below are a subset of all the commands available but they should enable any type of movement to be controlled. There are other commands available to perform other functions (for example for setting digital outputs) but these are not relevant for most applications that control movement. Some of these will nevertheless be listed here for completeness.

Where the same command has the SET/REQ/GET versions, the summary below only shows the SET version. The detailed description of the command lists the other versions.

Generic commands:

MGMSG_MOD_IDENTIFY	0x0223	Identify
MGMSG_HW_REQ_INFO	0x0005	Hardware information
MGMSG_MOD_SET_CHANENABLESTATE	0x0210	Enable channel

Move parameter setup commands:

MGMSG_MOT_SET_POSCOUNTER	0x0410	Set position counter
MGMSG_MOT_SET_VELPARAMS	0x0414	Set velocity parameters
MGMSG_MOT_SET_JOGPARAMS	0x0416	Set jogging parameters
MGMSG_MOT_SET_GENMOVEPARAMS	0x043A	Set general move parameters
MGMSG_MOT_SET_MOVERELPARAMS	0x0445	Set relative move parameters
MGMSG_MOT_SET_MOVEABSPARAMS	0x0450	Set absolute move parameters
MGMSG_MOT_SET_HOMEPARAMS	0x0442	Set parameters for homing

Move initiating commands:

MGMSG_MOT_MOVE_HOME	0x0443	Initiate homing
MGMSG_MOT_MOVE_RELATIVE	0x0448	Move relative
MGMSG_MOT_MOVE_ABSOLUTE	0x0453	Move absolute
MGMSG_MOT_MOVE_JOG	0x046A	Jog
MGMSG_MOT_MOVE_STOP	0x0465	Stop movement

Status update message related commands:

MGMSG_HW_START_UPDATEMSGS	0x0011	Start sending update messages
MGMSG_HW_STOP_UPDATEMSGS	0x0011	Stop sending update messages
MGMSG_MOT_REQ_DCSTATUSUPDATE	0x0490	Send status update
MGMSG_MOT_REQ_STATUSBITS	0x0429	Send status bits only

Error messages:

MGMSG_HW_RESPONSE	0x0080	Short error message
MGMSG_HW_RICHRESPONSE	0x0081	Verbose error message

The following sections detail the messages used for controller operations. Note that the source and destination fields are not filled in as these vary depending on the originator and target of the message.

7.1 Scaling Factors

To convert between the position and encoder counters in the stage being driven, and real world units, e.g. mm, the system uses certain conversion factors. These conversion factors differ depending on the stage being driven and the controller being used.

MLS203 and BBDxxx

Position:	1 mm equals 20,000 PMD units
Velocity:	1 mm / sec equals 134218 PMD units
Acceleration:	1 mm /sec^2 equals 13.7439 PMD units.
Jerk:	1 mm / sec^3 equals 92.2337 PMD “jerk” units

Please see the examples in each command description for details on using these conversion factors.

Generic System Control Messages

Introduction

The messages described here are either system control messages, or else generic messages which apply to several or all controller types. Please see the list of controller specific commands for details on applicability to a specific controller type.

MGMSG_MOD_IDENTIFY 0x0223

Function: Instruct hardware unit to identify itself (by flashing its front panel LEDs).

Command structure (6 bytes):

0	1	2	3	4	5
header only					
23	02	00	*00*	*d*	*s*

Example: Identify controller #1 (i.e. bay 0 of the TDC001 controller) by flashing its front panel LED.

TX 23, 02, 00, 00, 21, 01

MGMSG_MOD_SET_CHANENABLESTATE 0x0210
MGMSG_MOD_REQ_CHANENABLESTATE 0x0211
MGMSG_MOD_GET_CHANENABLESTATE 0x0212

Function Sent to enable or disable the specified drive channel.

SET:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
10	02	Chan Ident	Enable State	d	s

Channel Idents

0x01	channel 1
0x02	channel 2

Enable States

0x01	enable channel
0x02	disable channel

For single channel controllers such as the BBD10X, TDC001, the Chan Ident byte is always set to CHAN1.

Note: Although the BBD102 is in fact a 2-channel controller, ‘channel’ in this sense means “motor output channel within this module”. Electrically, the BBD102 is a bay system, with two bays, each of them being a single channel controller, so only one channel can be addressed. There are controllers in the Thorlabs product range which indeed have multiple output channels (for example the MST601 module) for which the channel ident is used to address a particular channel.

Example: Enable the motor channel in bay 2

TX 10, 02, 01, 01, 22, 01

REQ:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
11	02	Chan Ident	0	d	s

As above, for single channel controllers such as the BBD10X, TDC001, the Chan Ident byte is always set to CHAN1.

GET:

Response structure (6 bytes):

0	1	2	3	4	5
header only					
12	02	Chan Ident	Enable State	d	s

The meaning of the parameter bytes “Chan Ident” and “Enable State” is the same as for the SET version of the commands.

MGMSG_HW_DISCONNECT 0x002H

Function: Sent by the hardware unit or host when either wants to disconnect from the Ethernet/USB bus.

REQ:

Command structure (6 bytes):

0	1	2	3	4	5
header only					
02	00	00	00	d	s

Example: Disconnect the BBD103 from the USB bus

TX 02, 00, 00, 00, 11, 00

MGMSG_HW_RESPONSE 0x0080

Function: Sent by the hardware unit if it encounters a fault, failure or warning condition. In normal operation the HW_Response message will not be fired. It is good programming practice to handle this message in case hardware problems occur.

REQ:

Command structure (6 bytes):

0	1	2	3	4	5
header only					
80	00	00	00	d	s

Example: The BBD103 unit has encountered an over current condition

TX 80, 00, 00, 00, 01, 11

MGMSG_HW_START_UPDATEMSGS 0x0011

Function: Sent to start status updates from the embedded controller. Status update messages contain information about the position and status of the controller (for example limit switch status, motion indication, etc). The messages will be sent by the controller periodically until it receives a STOP STATUS UPDATE MESSAGES command. In applications where spontaneous messages (i.e. messages which are not received as a response to a specific command) must be avoided the same information can also be obtained by using the relevant GET_STATUTSUPDATES function.

Command structure (6 bytes):

0	1	2	3	4	5
header only					
11	00	Update Rate	Unused	d	s

The first data byte can be used to specify the update rate with which status updates are received from the controller. However, the parameter is ignored for the BBD101/102/103 controllers and the update rate is fixed at 10 regardless of the parameter sent.

REQUEST: **N/A**

MGMSG_HW_STOP_UPDATEMSGS 0x0012

Function: Sent to stop status updates from the controller – usually called by a client application when it is shutting down, to instruct the controller to turn off status updates to prevent USB buffer overflows on the PC.

SET:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
12	00	00	00	d	s

REQUEST: **N/A**
GET: **N/A**

MGMSG_HW_REQ_INFO 0x05H
MGMSG_HW_GET_INFO 0x06H

Function: Sent to request hardware information from the controller.

REQ:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
05	00	00	00	d	s

Example: Request hardware info from controller #1

TX 05, 00, 00, 00, 11, 01

GET:
Response structure (90 bytes):
6 byte header followed by 84 byte (0x54) data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
header						*data*									
06	00	54	00	d|	s	<-Serial Number >				<--------Model Number------->					

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
data															
<Model> No		<Type>		<--Software--> Version >				<------------------Notes------------------>							

32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
data															
<-- Notes --->															

48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
data															
<-- Notes --->															

64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
data															
<-- Notes --->															

80	81	82	83	84	85	86	87	88	89
data									
<--------------------Notes ------------------------>								<-nchs-->	

Data structure:

field	description	format
serial number	unique 8-digit serial number	long
model number	alphanumeric model number	char[8]
type	hardware type: 45 = multi-channel controller motherboard 44 = brushless DC controller	word

software version	software version
	byte[20] = minor revision number
	byte[21] = interim revision number
	byte[22] = major revision number
	byte[23] = unused
		byte[4]

notes	arbitrary alphanumeric information string	char[64]
nchs	number of channels	word

Example: Returned hardware info from controller #1

RX 06, 00, 54, 00, 81, 22, 89, 53, 9A, 05, 49, 4F, 4E, 30, 30, 31, 20, 00, 2C, 00, 02, 01, 39, 00, 42, 72, 75, 73, 68, 6C, 65, 73, 73, 20, 44, 43, 20, 4D, 6F, 74, 6F, 72, 20, 49, 4F, 4E, 20, 44, 72, 69, 76, 65, 00, 00…, 11, 00, 01, 00, 00, 00, 01, 00

Header: *06, 00, 54, 00, 81, 22*: Get Info, 54H (84) byte data packet, Motor Channel 2.
Serial Number: 89, 53, 9A, 05: 94000009
Model Number: 49, 4F, 4E, 30, 30, 31, 20, 00: ION001
Type*: 2C, 00:* 44 – Brushless DC Controller Card
Software Version: 02, 01, 39, 00: 3735810 (ION embedded code version)
Notes: 42, 72, 75, 73, 68, 6C, 65, 73, 73, 20, 44, 43, 20, 4D, 6F, 74, 6F, 72, 20, 49, 4F, 4E, 20, 44, 72, 69, 76, 65, 00…: BRUSHLESS DC MOTOR ION DRIVE…..
No Chan: 01, 00: 1 active channel

MGMSG_RACK_REQ_BAYUSED 0x060
MGMSG_RACK_GET_BAYUSED 0x061

Function: Sent to determine whether the specified bay in the controller is occupied.

REQ:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
60	00	Bay Ident	00	d	s

Bay Idents

0x01	Bay 1
0x02	Bay 2 to
0x09	Bay 10

Example: Is controller bay #1 (i.e. bay 0) occupied

TX 06, 00, 00, 00, 11, 01

GET:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
61	00	Bay Ident	Bay State	d	s

Bay Idents

0x01	Bay 1
0x02	Bay 2 to
0x09	Bay 10

Bay States

0x01	Bay Occupied
0x02	Bay Empty (Unused)

Example: Controller bay #1 (i.e. bay 0) is occupied

RX 06, 00, 00, 01, 11, 01

MGMSG_HUB_REQ_BAYUSED 0x065
MGMSG_HUB_GET_BAYUSED 0x066

Function: Sent to determine which bay a specific T-Cube is fitted.

REQ:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
65	00	00	00	d	s

TX 65, 00, 00, 00, 50, 01

GET:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
66	00	Bay Ident	00	d	s

Bay Idents

-0x01	T-Cube being standalone, i.e. off the hub.
0x00	T-Cube on hub, but bay unknown
0x01	Bay 1
0x02	Bay 2 to
0x06	Bay 6

Example: Which hub bay is the T-Cube unit fitted

TX 66, 00, 06, 00, 01, 50

MGMSG_RACK_REQ_STATUSBITS 0x0226
MGMSG_RACK_GET_STATUSBITS 0x0227

This method is applicable only to the MMR modular rack, and 2- and 3-channel card slot type controllers such as the BSC103 and BPC202.

Function: The USER IO connector on the rear panel of these units exposes a number of digital inputs. This function returns a number of status flags pertaining to the status of the inputs on the rack modules, or the motherboard of the controller unit hosting the single channel controller card.

These flags are returned in a single 32 bit integer parameter and can provide additional useful status information for client application development. The individual bits (flags) of the 32 bit integer value are described below.

REQUEST:

Command structure (6 bytes):

0	1	2	3	4	5
header only					
28	02	Status Bits	00	d	s

GET:

Response structure (10 bytes)

6 byte header followed by 4 byte data packet as follows:

0	1	2	3	4	5	7	8	9	10
header						*Data*			
27	02	04	00	d|	s	StatusBits			

Data Structure:

field	description	format
StatusBits	The status bits for the associated controller channel. The meaning of the individual bits (flags) of the 32 bit integer value will depend on the controller and are described in the following table.	dword

Hex Value	Bit Number	Description
0x00000001	1	Digital output 1 state (1 - logic high, 0 - logic low).
0x00000002	2	Digital output 2 state (1 - logic high, 0 - logic low).
0x00000004	3	Digital output 3 state (1 - logic high, 0 - logic low).
0x00000008	4	Digital output 4 state (1 - logic high, 0 - logic low).

Example: With destination being 0x11 (motherboard – see Introduction) and bay being bay 1, slot 2 (0x22)

TX 27, 02, 04, 00, 01, 22, 00, 00, 00, 00

Header: 27, 02, 04, 00, 01, 22: GetStatusBits, 04 byte data packet, bay 1 slot 2.

MGMSG_RACK_SET_DIGOUTPUTS 0x0228
MGMSG_RACK_REQ_DIGOUTPUTS 0x0229
MGMSG_RACK_GET_DIGOUTPUTS 0x0230

This method is applicable only to the MMR rack modules, and 2- and 3-channel card slot type controllers such as the BSC103 and BPC202.

Function: The USER IO connector on the rear panel of these units exposes a number of digital outputs. These functions set and return the status of the outputs on the rack modules, or the motherboard of the controller unit hosting the single channel controller card.
These flags are returned in a single 32 bit integer parameter and can provide additional useful status information for client application development. The individual bits (flags) of the 32 bit integer value are described below.

SET:
Data structure (6 bytes)

0	1	2	3	4	5
header only					
28	02	Dig OP	00	d	s

Hex Value	Bit Number	Description
0x00000001	1	Digital input 1 state (1 - logic high, 0 - logic low).
0x00000002	2	Digital input 2 state (1 - logic high, 0 - logic low).
0x00000004	3	Digital input 3 state (1 - logic high, 0 - logic low).
0x00000008	4	Digital input 4 state (1 - logic high, 0 - logic low).

Example: With destination being 0x11 (motherboard – see Introduction) and bay being bay 1, slot 2 (0x22), set Digital output 1 high

TX 28, 02, 01, 22, 11, 01,

Header: 28, 02, 01, 22, 11, 01: SetDigOutputs, 01 OP1 High, bay 1 slot 2, d=motherboard, s=PC.

REQUEST:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
29	02	00	00	d	s

GET:
Response structure (6 bytes)

0	1	2	3	4	5
header only					
30	02	00	00	d	s

See SET above for structure

Motor Control Messages

Introduction

The ‘Motor’ messages provide the functionality required for a client application to control one or more of the Thorlabs series of motor controller units. This range of motor controllers covers DC servo and stepper drivers in a variety of formats including compact Cube type controllers, benchtop units and 19" rack based modular drivers. Note for ease of description, the TSC001 T-Cube Solenoid Controller is considered here as a motor controller. The list of controllers covered by the motor messages includes:

BSC001 – 1 Channel Benchtop Stepper Driver
BSC002 – 2 Channel Benchtop Stepper Driver
BMS001 – 1 Channel Benchtop Low Power Stepper Driver
BMS002 – 2 Channel Benchtop Low Power Stepper Driver
MST601 – 2 Channel Modular Stepper Driver
BSC101 – 1 Channel Benchtop Stepper Driver (2006 onwards)
BSC102 – 2 Channel Benchtop Stepper Driver (2006 onwards)
BSC103 – 3 Channel Benchtop Stepper Driver (2006 onwards)
BBD101 – 1 Channel Benchtop Brushless DC Motor Driver
BBD102 – 2 Channel Benchtop Brushless DC Motor Driver
BBD103 – 3 Channel Benchtop Brushless DC Motor Driver
BBD201 – 1 Channel Benchtop Brushless DC Motor Driver
BBD202 – 2 Channel Benchtop Brushless DC Motor Driver
BBD203 – 3 Channel Benchtop Brushless DC Motor Driver
OST001 – 1 Channel Cube Stepper Driver
ODC001 – 1 Channel Cube DC Servo Driver
TST001 – 1 Channel T-Cube Stepper Driver
TDC001 – 1 Channel T-Cube DC Servo Driver
TSC001 – 1 Channel T-Cube Solenoid Driver

The motor messages can be used to perform activities such as homing stages, absolute and relative moves, changing velocity profile settings and operation of the solenoid state (TSC001 T-Cube). With a few exceptions, these messages are generic and apply equally to both single and dual channel units.
Where applicable, the target channel is identified in the Chan Ident parameter and on single channel units, this must be set to CHAN1_ID. On dual channel units, this can be set to CHAN1_ID, CHAN2_ID or CHANBOTH_ID as required.

For details on the operation of the motor controller, and information on the principles of operation, refer to the handbook supplied with the unit.

MGMSG_MOT_SET_POSCOUNTER 0x0410
MGMSG_MOT_REQ_POSCOUNTER 0x0411
MGMSG_MOT_GET_POSCOUNTER 0x0412

Function: Used to set the 'live' position count in the controller. In general, this command is not normally used for the brushless DC controller family. Instead, the stage is homed immediately after power-up (at this stage the position is unknown as the stage is free to move when the power is off); and after the homing process is completed the position counter is automatically updated to show the actual position. From this point onwards the position counter always shows the actual absolute position.

SET:
Command structure (12 bytes)
6 byte header followed by 6 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
10	04	06	00	d|	s	Chan Ident		Position			

Data Structure:

field	description	format
Chan Ident	The channel being addressed	word
Position	The new value of the position counter as a 32-bit signed integer, encoded in the Intel format. The scaling between real time values and this parameter is detailed in section 7.1.	long

Example: MLS203 and BBD102: Set the position counter for channel 2 to 10.0 mm

TX 10, 04, 06, 00, A2, 01, 01, 00, 04, 0D, 03, 00

Header: 10, 04, 06, 00, A2, 01: SetPosCounter, 06 byte data packet, Channel 2.
Chan Ident: 01, 00: Channel 1 (always set to 1 for TDC001)
Position: 04, 0D, 03, 00: Set Counter to 10 mm (10 x 20,000)

REQUEST:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
11	04	Chan Ident	00	d	s

GET:
Response structure (12 bytes)
6 byte header followed by 6 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
12	04	06	00	d|	s	Chan Ident		Position			

For structure see SET message above.

MGMSG_MOT_SET_ENCCOUNTER 0x0409
MGMSG_MOT_REQ_ENCCOUNTER 0x040A
MGMSG_MOT_GET_ENCCOUNTER 0x040B

Function: Used to set the encoder count in the controller. In general, this command is not normally used for the brushless DC controller family. Instead, the stage is homed immediately after power-up (at this stage the position is unknown as the stage is free to move when the power is off); and after the homing process is completed the position counter is automatically updated to show the actual position. From this point onwards the position counter always shows the actual absolute position.

SET:
Command structure (12 bytes)
6 byte header followed by 6 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
09	04	06	00	d|	s	Chan Ident		Encoder Count			

Data Structure:

field	description	format
Chan Ident	The channel being addressed	word
Encoder Count	The new value of the encoder counter as a 32-bit signed integer, encoded in the Intel format. The scaling between real time values and this parameter is detailed in section 7.1.	long

Example: MLS203 and BBD102: Set the encoder counter for channel 2 to 10.0 mm

TX 09, 04, 06, 00, A2, 01, 01, 00, 04, 0D, 03, 00

Header: 09, 04, 06, 00, A2, 01: SetEncCounter, 06 byte data packet, Channel 2.
Chan Ident: 01, 00: Channel 1 (always set to 1 for TDC001)
Position: 04, 0D, 03, 00: Set Counter to 10 mm (10 x 20,000)

REQUEST:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
11	04	Chan Ident	00	d	s

GET:
Response structure (12 bytes)
6 byte header followed by 6 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
0B	04	06	00	d|	s	Chan Ident		Encoder Count			

For structure see SET message above.

MGMSG_MOT_SET_VELPARAMS 0x0413
MGMSG_MOT_REQ_VELPARAMS 0x0414
MGMSG_MOT_GET_VELPARAMS 0x0415

Function: Used to set the trapezoidal velocity parameters for the specified motor channel, in encoder counts/sec for velocity or encoder counts/sec/sec for acceleration.
For stepper controllers the position steps are in micro-steps and for DC servo controller in encoder counts.

SET:
Command structure (20 bytes)
6 byte header followed by 14 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
13	04	0E	00	d|	s	Chan Ident		Min Velocity			

12	13	14	15	16	17	18	19
Data							
Acceleration				Max Velocity			

Data Structure:

field	description	format
Chan Ident	The channel being addressed	word
Minimum (Start) Vel	The minimum (start) velocity in encoder counts/sec Currently, this 4 byte value is always zero	long
Acceleration	The acceleration in encoder counts /sec/sec. 4 byte unsigned long value. The scaling between real time values and this parameter is detailed in section 7.1.	long
Maximum Vel	The maximum (final) velocity in encoder counts /sec. 4 byte unsigned long value. The scaling between real time values and this parameter is detailed in section 7.1.	long

Example: MLS203 and BBD102: Set the trapezoidal velocity parameters for chan 2 as follows:

Min Vel: zero
Acceleration: 10 mm/sec/sec
Max Vel: 99 mm/sec

TX 13, 04, 0E, 00, A2, 01, 01, 00, 00, 00, 00, 00, B0, 35, 00, 00, CD, CC, CC, 00

Header: 13, 04, 0E, 00, A2, 01: Set Vel Params, 0EH (14) byte data packet, Channel 2.
Chan Ident: 01, 00: Channel 1 (always set to 1 for TDC001)
Min Vel: *00, 00, 00, 00:* Set min velocity to zero
Accel: *89, 00, 00, 00:* Set acceleration to 10 mm/sec/sec (13.744 x 10)
Max Vel: *9E, C0, CA, 00:* Set max velocity to 99 mm/sec (134218 x 99)

REQUEST:

Command structure (6 bytes):

0	1	2	3	4	5
header only					
14	04	Chan Ident	00	d	s

GET:

Response structure (20 bytes)
6 byte header followed by 14 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
15	04	0E	00	d|	s	Chan Ident		Min Velocity			

12	13	14	15	16	17	18	19
Data							
Acceleration				Max Velocity			

For structure see SET message above.

MGMSG_MOT_SET_JOGPARAMS 0x0416
MGMSG_MOT_REQ_JOGPARAMS 0x0417
MGMSG_MOT_GET_JOGPARAMS 0x0418

Function: Used to set the velocity jog parameters for the specified motor channel, in position steps/sec for velocity or position steps/sec/sec for acceleration.

SET:

Command structure (28 bytes)

6 byte header followed by 22 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
16	04	16	00	d|	s	Chan Ident		Jog Mode		Jog Step Size	

12	13	14	15	16	17	18	19	20	21
Data									
Jog Step Size		Jog Min Velocity				Jog Acceleration			

22	23	24	25	26	27
Data					
Jog Max Velocity			Stop Mode		

Data Structure:

field	description	format
Chan Ident	The channel being addressed	word
Jog Mode	This 2 byte value can be 1 for continuous jogging or 2 for single step jogging. In continuous jogging mode the movement continues for as long as the jogging trigger (the jogging button on the GUI or an external signal) is being active. In single step mode triggering jogging initiates a single move whose step size is defined as the next parameter (see below).	word
Jog Step Size	The jog step size in encoder counts. The scaling between real time values and this parameter is detailed in section 7.1.	long
Jog Min Velocity	The minimum (start) velocity in encoder counts /sec. Currently, this 4 byte value is always zero.	long
Jog Acceleration	The acceleration in encoder counts /sec/sec The scaling between real time values and this parameter is detailed in section 7.1.	long
Jog Max Velocity	The maximum (final) velocity in encoder counts /sec. The scaling between real time values and this parameter is detailed in section 7.1.	long
Jog Stop Mode	The stop mode. This 16 bit word can be 1 for immediate (abrupt) stop or 2 for profiled stop (with controlled deceleration).	word

Example: MLS203 and BBD102: Set the jog parameters for channel 2 as follows:
Jog Mode: Continuous
Jog Step Size:0.05 mm
Jog Min Vel: Zero
Jog Accel: 10 mm/sec/sec
Jog Max Vel: 99 mm/sec
Jog Stop Mode: Profiled

TX 16, 04, 16, 00, A2, 01, 01, 00, 01, 00, E8, 03, 00, 00, 00, 00, 00, 00, B0,35, 00, 00, CD, CC, CC, 00, 02, 00

Header: 16, 04, 16, 00, A2, 01: Set Jog Params, 16H (28) byte data packet, Channel 2.
Chan Ident: 01, 00: Channel 1 (always set to 1 for TDC001)
Jog Mode: *01,00,:* Set jog mode to ‘continuous’
Jog Step Size: *E8, 03, 00, 00:* Set jog step size to 0.05 mm (1,000 encoder counts).
Jog Min Vel: 00, 00, 00, 00: Set min jog velocity to zero
Jog Accel: *89, 00, 00, 00:* Set acceleration to 10 mm/sec/sec (13.744 x 10)
Jog Max Vel: *9E, C0, CA, 00:* Set max velocity to 99 mm/sec (134218 x 99)
Jog Stop Mode: 02, 00: Set jog stop mode to ‘Profiled Stop’.

REQUEST:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
17	04	Chan Ident	00	d	s

GET:
Response structure (28 bytes)
6 byte header followed by 22 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
18	04	16	00	d|	s	Chan Ident		Jog Mode		Jog Step Size	

12	13	14	15	16	17	18	19	20	21
Data									
Jog Step Size		Jog Min Velocity				Jog Acceleration			

22	23	24	25	26	27
Data					
Jog Max Velocity				Stop Mode	

For structure see SET message above.

MGMSG_MOT_SET_GENMOVEPARAMS 0x043A
MGMSG_MOT_REQ_GENMOVEPARAMS 0x043B
MGMSG_MOT_GET_GENMOVEPARAMS 0x043C

Function: Used to set the general move parameters for the specified motor channel. At this time this refers specifically to the backlash settings.

SET:
Command structure (12 bytes)
6 byte header followed by 6 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
3A	04	06	00	d|	s	Chan Ident		Backlash Distance			

Data Structure:

field	description	format
Chan Ident	The channel being addressed	word
Backlash Distance	The value of the backlash distance as a 4 byte signed integer, which specifies the relative distance in position counts. The scaling between real time values and this parameter is detailed in section 7.1.	long

Example: MLS203 and BBD102: Set the backlash distance for chan 2 to 1 mm:

TX 3A, 04, 06, 00, A2, 01, 01, 00, 20, 4E, 00, 00,

Header: 3A, 04, 06, 00, A2, 01: SetGenMoveParams, 06 byte data packet, Channel 2.
Chan Ident: 01, 00: Channel 1 (always set to 1 for TDC001)
Backlash Dist: *20, 4E, 00, 00:* Set backlash distance to 1 mm (20,000 encoder counts).

REQUEST:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
3B	04	Chan Ident	00	d	s

GET:
Response structure (12 bytes)
6 byte header followed by 6 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
3C	04	06	00	d|	s	Chan Ident		Backlash Distance			

For structure see SET message above.

MGMSG_MOT_SET_MOVERELPARAMS 0x0445
MGMSG_MOT_REQ_MOVERELPARAMS 0x0446
MGMSG_MOT_GET_MOVERELPARAMS 0x0447

Function: Used to set the relative move parameters for the specified motor channel. The only significant parameter at this time is the relative move distance itself. This gets stored by the controller and is used the next time a relative move is initiated.

SET:
Command structure (12 bytes)
6 byte header followed by 6 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
45	04	06	00	d|	s	Chan Ident		Relative Distance			

Data Structure:

field	description	format
Chan Ident	The channel being addressed	word
Relative Distance	The distance to move. This is a 4 byte signed integer that specifies the relative distance in position encoder counts. The scaling between real time values and this parameter is detailed in section 7.1.	long

Example: MLS203 and BBD102: Set the relative move distance for chan 2 to 10 mm:

TX 45, 04, 06, 00, A2, 01, 01, 00, 40, 0D, 03, 00,

Header: 45, 04, 06, 00, A2, 01: SetMoveRelParams, 06 byte data packet, Channel 2.
Chan Ident: 01, 00: Channel 1 (always set to 1 for TDC001)
Rel Dist: *40, 0D, 03, 00:* Set relative move distance to 10 mm (10 x 20,000 encoder counts).

REQUEST:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
46	04	Chan Ident	00	d	s

GET:
Response structure (12 bytes)
6 byte header followed by 6 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
47	04	06	00	d|	s	Chan Ident		Relative Distance			

For structure see SET message above.

MGMSG_MOT_SET_MOVEABSPARAMS 0x0450
MGMSG_MOT_REQ_MOVEABSPARAMS 0x0451
MGMSG_MOT_GET_MOVEABSPARAMS 0x0452

Function: Used to set the absolute move parameters for the specified motor channel. The only significant parameter at this time is the absolute move position itself. This gets stored by the controller and is used the next time an absolute move is initiated.

SET:
Command structure (12 bytes)
6 byte header followed by 6 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
50	04	06	00	d|	s	Chan Ident		Absolute Position			

Data Structure:

field	description	format
Chan Ident	The channel being addressed	word
Absolute Position	The absolute position to move. This is a 4 byte signed integer that specifies the absolute position in position encoder counts. The scaling between real time values and this parameter is detailed in section 7.1.	long

Example: MLS203 and BBD102: Set the absolute move position for chan 2 to 10 mm:

TX 50, 04, 06, 00, A2, 01, 01, 00, 40, 0D, 03, 00,

Header: 50, 04, 06, 00, A2, 01: SetMoveAbsParams, 06 byte data packet, Channel 2.
Chan Ident: 01, 00: Channel 1 (always set to 1 for TDC001)
Abs Pos: *40, 0D, 03, 00:* Set absolute move position to 10 mm (200,000 encoder counts).

REQUEST:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
51	04	Chan Ident	00	d	s

GET:
Response structure (12 bytes)
6 byte header followed by 6 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
52	04	06	00	d|	s	Chan Ident		Absolute Position			

For structure see SET message above.

MGMSG_MOT_SET_HOMEPARAMS 0x0440
MGMSG_MOT_REQ_HOMEPARAMS 0x0441
MGMSG_MOT_GET_HOMEPARAMS 0x0442

Function: Used to set the home parameters for the specified motor channel. These parameters are stage specific and for the MLS203 stage implementation the only parameter that can be changed is the homing velocity.

SET:

Command structure (20 bytes)
6 byte header followed by 14 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
40	04	0E	00	d|	s	Chan Ident		Home Dir		Limit Switch	

12	13	14	15	16	17	18	19
Data							
Home Velocity				Offset Distance			

Data Structure:

field	description	format
Chan Ident	The channel being addressed	word
Home Direction	Ignored in this implementation. Homing direction is always positive.	word
Limit Switch	Ignored in this implementation. The limit switches are not used for homing.	word
Home Velocity	The homing velocity. A 4 byte unsigned long value. The scaling between real time values and this parameter is detailed in section 7.1.	long
Offset Distance	Not used in this implementation.	long

Example: MLS203 and BBD102: Set the home parameters for chan 2 as follows:
Home Direction: Not used (always positive).
Limit Switch: Not used
Home Vel: 24 mm/sec
Offset Dist: Not used.

TX 40, 04, 0E, 00, A2, 01, 01, 00, 00, 00, 00, 00, 33. 33, 33, 00, 00, 00, 00, 00

Header: 40, 04, 0E, 00, A2, 01: SetHomeParams, 14 byte data packet, Channel 2.
Chan Ident: 01, 00: Channel 1 (always set to 1 for TDC001)
Home Direction: *00, 00:* Not Applicable
Limit Switch: 00, 00: Not Applicable
Home Velocity: 33, 33, 33, 00: 24 mm/sec (3355443/134218)
Offset Distance: 00, 00, 00, 00: Not used

REQUEST:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
41	04	Chan Ident	00	d	s

GET:
Response structure (20 bytes)
6 byte header followed by 14 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
42	04	0E	00	d|	s	Chan Ident		Home Dir		Limit Switch	

12	13	14	15	16	17	18	19
Data							
Home Velocity				Offset Distance			

For structure see SET message above.

MGMSG_MOT_SET_LIMSWITCHPARAMS 0x0423
MGMSG_MOT_REQ_LIMSWITCHPARAMS 0x0424
MGMSG_MOT_GET_LIMSWITCHPARAMS 0x0425

These functions are not applicable to BBD10x units

Function: Used to set the limit switch parameters for the specified motor channel.

SET:
Command structure (22 bytes)
6 byte header followed by 16 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
23	04	10	00	d|	s	Chan Ident		CW Hardlimit		CCW Hardlimit	

12	13	14	15	16	17	18	19	20	21
Data									
CW Soft Limit				CCW Soft Limit				Limit Mode	

Data Structure:

field	description	format
Chan Ident	The channel being addressed	word
CW Hard Limit	The operation of the Clockwise hardware limit switch when contact is made. 0x01 Ignore switch or switch not present. 0x02 Switch makes on contact. 0x03 Switch breaks on contact. 0x04 Switch makes on contact - only used for homes (e.g. limit switched rotation stages). 0x05 Switch breaks on contact - only used for homes (e.g. limit switched rotations stages). 0x06 For PMD based brushless servo controllers only - uses index mark for homing. Note. Set upper bit to swap CW and CCW limit switches in code. Both CWHardLimit and CCWHardLimit structure members will have the upper bit set when limit switches have been physically swapped. 0x80 // bitwise OR'd with one of the settings above.	word
CCW Hard Limit	The operation of the Counter Clockwise hardware limit switch when contact is made.	word
CW Soft Limit	Clockwise software limit in position steps. A 32 bit unsigned long value, the scaling factor between real time values and this parameter is 1 mm is equivalent to 134218. For example, to set the clockwise software limit switch to 100 mm, send a value of 13421800.	long
CCW Soft Limit	Counter Clockwise software limit in position steps (scaling as for CW limit).	long
Software	Software limit switch mode	word

Limit Mode	0x01 Ignore Limit 0x02 Stop Immediate at Limit 0x03 Profiled Stop at limit 0x80 Rotation Stage Limit (bitwise OR'd with one of the settings above)	

Example: Set the limit switch parameters for chan 2 as follows:
CW Hard Limit – switch makes.
CCW Hard Limit - switch makes
CW Soft Limit – set to 100 mm
CCW Soft Limit - .set to 0 mm
Software Limit Mode – Profiled Stop

TX 23, 04, 10, 00, A2, 01, 01, 00, 02, 00, 02, 00, E8. CC, CC, 00, 00, 00, 00, 00, 03, 00

Header: 23, 04, 10, 00, A2, 01: SetLimSwitchParams, 16 byte data packet, Channel 2.
Chan Ident: 01, 00: Channel 1 (always set to 1 for TDC001)
CW Hard Limit: *02, 00:* Switch Makes
CCW Hard Limit: 02, 00: Switch Makes
CW Soft Limit: E8, CC, CC, 00: 100 mm (13421800/134218)
CCW Soft Limit: 00, 00, 00, 00: 0 mm
Soft Limit Mode: 03, 00: Profiled Stop at Limit

REQUEST:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
24	04	Chan Ident	00	d	s

GET:
Response structure (20 bytes)
6 byte header followed by 16 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
25	04	10	00	d|	s	Chan Ident		CW Hardlimit		CCW Hardlimit	

12	13	14	15	16	17	18	19	20	21
Data									
CW Soft Limit				CCW Soft Limit				Limit Mode	

For structure see SET message above.

MGMSG_MOT_MOVE_HOME 0x0443
MGMSG_MOT_MOVE_HOMED 0x0444

Function: Sent to start a home move sequence on the specified motor channel (in accordance with the home parameters above).

TX structure (6 bytes):

0	1	2	3	4	5
header only					
43	04	Chan Ident	0x	d	s

Example: Home the motor channel in bay 2

TX 43, 04, 01, 00, 22, 01

HOMED:

Function: No response on initial message, but upon completion of home sequence controller sends a "homing completed" message:

RX structure (6 bytes):

0	1	2	3	4	5
header only					
44	04	Chan Ident	0x	d	s

Example: The motor channel in bay 2 has been homed

RX 44, 04, 01, 00, 01, 22

MGMSG_MOT_MOVE_RELATIVE 0x0448

Function: This command can be used to start a relative move on the specified motor channel (using the relative move distance parameter above). There are two versions of this command: a shorter (6-byte header only) version and a longer (6 byte header plus 6 data bytes) version. When the first one is used, the relative distance parameter used for the move will be the parameter sent previously by a MGMSG_MOT_SET_MOVERELPARAMS command. If the longer version of the command is used, the relative distance is encoded in the data packet that follows the header.

Short version:

TX structure (6 bytes):

0	1	2	3	4	5
header only					
48	04	Chan Ident	0x	d	s

Example: Move the motor associated with channel 2 by 10 mm. (10 mm was previously set in the MGMSG_ MOT_SET_MOVERELPARAMS method).

TX 48, 04, 01, 00, 22, 01

Long version:
The alternative way of using this command is by appending the relative move params structure (MOT_SET_MOVERELPARAMS) to this message header.

Command structure (12 bytes)
6 byte header followed by 6 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
48	04	06	00	d|	s	Chan Ident		Relative Distance			

Data Structure:

field	description	format
Chan Ident	The channel being addressed	Word
Relative Distance	The distance to move. This is a 4 byte signed integer that specifies the relative distance in position encoder counts. In the BBD10X series controllers the encoder resolution is 20,000 counts per mm, therefore to set a relative move distance of 1 mm, set this parameter to 20,000 (twenty thousand).	Long

Example: Move the motor associated with chan 2 by 10 mm:

TX 48, 04, 06, 00, A2, 01, 01, 00, 40, 0D, 03, 00,

Header: 45, 04, 06, 00, A2, 01: MoveRelative, 06 byte data packet, Channel 2.
Chan Ident: 01, 00: Channel 1 (always set to 1 for TDC001)
Rel Dist: *40, 0D, 03, 00:* Set absolute move distance to 10 mm (200,000 encoder counts).

Upon completion of the relative move the controller sends a Move Completed message as described following.

MGMSG_MOT_MOVE_COMPLETED 0x0464

Function: No response on initial message, but upon completion of the relative or absolute move sequence, the controller sends a “move completed” message:

RX structure (20 bytes):

0	1	2	3	4	5
header only					
64	04	Chan Ident	0x	d|	s

Followed by a 14-byte data packet described by the same status structures (i.e. MOTSTATUS and MOTDCSTATUS) described in the STATUS UPDATES section that follows.

MGMSG_MOT_MOVE_ABSOLUTE 0x0453

Function: Used to start an absolute move on the specified motor channel (using the absolute move position parameter above). As previously described in the “MOVE RELATIVE” command, there are two versions of this command: a shorter (6-byte header only) version and a longer (6 byte header plus 6 data bytes) version. When the first one is used, the absolute move position parameter used for the move will be the parameter sent previously by a MGMSG_MOT_SET_MOVEABSPARAMS command. If the longer version of the command is used, the absolute position is encoded in the data packet that follows the header.

Short version:

TX structure (6 bytes):

0	1	2	3	4	5
header only					
53	04	Chan Ident	0x	d	s

Example: Move the motor associated with channel 2 to 10 mm. (10 mm was previously set in the MGMSG_ MOT_SET_MOVEABSPARAMS method).

TX 53, 04, 01, 00, 22, 01

Long version:

The alternative way of using this command by appending the absolute move params structure (MOTABSMOVEPARAMS) to this message header.

Command structure (12 bytes)
6 byte header followed by 6 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
53	04	06	00	d|	s	Chan Ident		Absolute Distance			

Data Structure:

field	description	format
Chan Ident	The channel being addressed	Word
Absolute Distance	The distance to move. This is a 4 byte signed integer that specifies the absolute distance in position encoder counts. In the BBD10X series controllers the encoder resolution is 20,000 counts per mm, therefore to set an absolute move distance of 100 mm, set this parameter to 2,000,000 (two million).	Long

Example: Move the motor associated with chan 2 to 10 mm:

TX 53, 04, 06, 00, A2, 01, 01, 00, 40, 0D, 03, 00,

Header: 45, 04, 06, 00, A2, 01: MoveAbsolute, 06 byte data packet, Channel 2.
Chan Ident: 01, 00: Channel 1 (always set to 1 for TDC001)
Abs Dist: *40, 0D, 03, 00:* Set the absolute move distance to 10 mm (200,000 encoder counts).

Upon completion of the absolute move the controller sends a Move Completed message as previously described.

MGMSG_MOT_MOVE_JOG 0x046A

Function: Sent to start a jog move on the specified motor channel.

TX structure (6 bytes):

0	1	2	3	4	5
header only					
6A	04	Chan Ident	Direction	d	s

Data Structure:

field	description	format
Chan Ident	The channel being addressed	word
Direction	The direction to Jog. Set this byte to 0x01 to jog forward, or to 0x02 to jog in the reverse direction.	word

Upon completion of the jog move the controller sends a Move Completed message as previously described.

MGMSG_MOT_MOVE_VELOCITY 0x0457

Function: This command can be used to start a move on the specified motor channel.
When this method is called, the motor will move continuously in the specified direction, using the velocity parameters set in the MGMSG_MOT_SET_MOVEVELPARAMS command until either a stop command (either StopImmediate or StopProfiled) is called, or a limit switch is reached.

TX structure (6 bytes):

0	1	2	3	4	5
header only					
57	04	Chan Ident	Direction	d	s

Data Structure:

field	**description**	**format**
Chan Ident	The channel being addressed	word
Direction	The direction to Jog. Set this byte to 0x01 to move forward, or to 0x02 to move in the reverse direction.	word

Upon completion of the move the controller sends a Move Completed message as previously described.

Example: Move the motor associated with channel 2 forwards.

TX 57, 04, 01, 01, 22, 01

MGMSG_MOT_MOVE_STOP 0x0465

Function: Sent to stop any type of motor move (relative, absolute, homing or move at velocity) on the specified motor channel.

TX structure (6 bytes):

0	1	2	3	4	5
header only					
65	04	Chan Ident	Stop Mode	d	s

Data Structure:

field	description	format
Chan Ident	The channel being addressed	word
Stop Mode	The stop mode defines either an immediate (abrupt) or profiles tops. Set this byte to 0x01 to stop immediately, or to 0x02 to stop in a controller (profiled) manner.	word

Upon completion of the stop move the controller sends a Move Stopped message as described following

MGMSG_MOT_MOVE_STOPPED 0x0466

Function: No response on initial message, but upon completion of the stop move, the controller sends a “move stopped” message:

RX structure (20 bytes):

0	1	2	3	4	5
header only					
66	04	0E	0x	d|	s

Followed by a 14-byte data packet described by the same status structures (i.e. MOTSTATUS and MOTDCSTATUS) described in the STATUS UPDATES section that follows.

MGMSG_MOT_SET_DCPIDPARAMS 0x04A0
MGMSG_MOT_REQ_DCPIDPARAMS 0x04A1
MGMSG_MOT_GET_DCPIDPARAMS 0x04A2

Function:

Used to set the position control loop parameters for the specified motor channel.

The motion processor within the controller uses a position control loop to determine the motor command output. The purpose of the position loop is to match the actual motor position and the demanded position. This is achieved by comparing the demanded position with the actual position to create a position error, which is then passed through a digital PID-type filter. The filtered value is the motor command output.

NOTE. These settings apply to LM628/629 based servo controllers (only TDC001 at this time). Refer to data sheet for National Semiconductor LM628/LM629 for further details on setting these PID related parameters.

SET:

Command structure (26 bytes)

6 byte header followed by 20 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
A0	04	14	00	d|	s	Chan Ident		Proportional			

12	13	14	15	16	17	18	19	20	21	22	23
Data											
Integral				Differential				Integral Limit			

24	25
Data	
FilterControl	

Data Structure:

field	description	format
Chan Ident	The channel being addressed	word
Proportional	The proportional gain. Together with the Integral and Differential, these terms determine the system response characteristics and accept values in the range 0 to 32767.	long
Integral	The integral gain. Together with the Proportional and Differential, these terms determine the system response characteristics and accept values in the range 0 to 32767.	long
Differential	The differential gain. Together with the Proportional and Integral, these terms determine the system response characteristics and accept values in the range 0 to 32767.	long
Integral Limit	The Integral Limit parameter is used to cap the value of the Integrator to prevent runaway of the integral sum at the output. It accepts values in the range 0 to 32767. If set to 0 then the integration term in the PID loop is ignored.	long
FilterControl	Identifies which of the above parameters are applied by	word

	setting the corresponding bit to '1'. By default, all parameters are applied, and this parameter is set to 0F (1111).	

Example: Set the PID parameters for TDC001 as follows:
Proportional: 65
Integral: 175
Differential: 600
Integral Limit: 20,000
FilCon: 15

TX A0, 04, 14, 00, D0, 01, 01, 00, 41, 00, AF, 00, 58, 02, 20, 4E, 00, 00, 0F, 00

Header: A0, 04, 14, 00, D0, 01: Set_DCPIDParams, 20 byte data packet, Generic USB Device.
Chan Ident: 01, 00: Channel 1 (always set to 1 for TDC001)
Proportional: *41, 00,:* Set the proportional term to 65
Integral: *AF, 00,:* Set the integral term to 175
Differential: 58, 02,: Set the differential term to 600
Integral Limit: 20, 4E, 00, 00,: Set the integral limit to 20,000
FilterControl: 0F, 00: Set all terms to active.

REQUEST:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
A0	04	Chan Ident	00	d	s

GET:
6 byte header followed by 20 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
A0	04	14	00	d|	s	Chan Ident		Proportional			

12	13	14	15	16	17	18	19	20	21	22	23
Data											
Integral				Differential				Integral Limit			

24	25
Data	
FilterControl	

For structure see Set message above.

MGMSG_MOT_SET_AVMODES 0x04B3
MGMSG_MOT_REQ_AVMODES 0x04B4
MGMSG_MOT_GET_AVMODES 0x04B5

Function: The LED on the control keypad can be configured to indicate certain driver states.
All modes are enabled by default. However, it is recognised that in a light sensitive environment, stray light from the LED could be undesirable. Therefore it is possible to enable selectively, one or all of the LED indicator modes described below by setting the appropriate value in the Mode Bits parameter.

SET:
Command structure (10 bytes)
6 byte header followed by 4 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9
header						*Data*			
B3	04	04	00	d|	s	Chan Ident		ModeBits	

Data Structure:

field	description	format
Chan Ident	The channel being addressed	word
ModeBits	The mode of operation for the LED is set according to the hex value entered in the mode bits. 1 LEDMODE_IDENT: The LED will flash when the 'Ident' message is sent. 2 LEDMODE_LIMITSWITCH: The LED will flash when the motor reaches a forward or reverse limit switch. 8 LEDMODE_MOVING: The LED is lit when the motor is moving.	word

Example: Set the LED to flash when the IDENT message is sent, and also when the motor is moving.

TX B3, 04, 04, 00, D0, 01, 01, 00, 09, 00,

Header: B3, 04, 04, 00, D0, 01: SetAVModes, 04 byte data packet, Generic USB Device.
Chan Ident: 01, 00: Channel 1 (always set to 1 for TDC001)
ModeBits: 09, 00 (i.e. 1 + 8)

Similarly, if the ModeBits parameter is set to '11' (1 + 2 + 8) all modes will be enabled.

REQUEST:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
11	04	Chan Ident	00	d	s

GET:

Response structure (10 bytes)
6 byte header followed by 4 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9
header						*Data*			
B5	04	04	00	d|	s	Chan Ident		ModeBits	

For structure see SET message above.

MGMSG_MOT_SET_POTPARAMS 0x04B0
MGMSG_MOT_REQ_POTPARAMS 0x04B1
MGMSG_MOT_GET_POTPARAMS 0x04B2

Function: The potentiometer slider on the control panel panel is sprung, such that when released it returns to it's central position. In this central position the motor is stationary. As the slider is moved away from the center, the motor begins to move; the speed of this movement increases as the slider deflection is increased. Bidirectional control of motor moves is possible by moving the slider in both directions. The speed of the motor increases by discrete amounts rather than continuously, as a function of slider deflection. These speed settings are defined by 4 pairs of parameters. Each pair specifies a pot deflection value (in the range 0 to 127) together with an associated velocity (set in encoder counts/sec) to be applied at or beyond that deflection. As each successive deflection is reached by moving the pot slider, the next velocity value is applied. These settings are applicable in either direction of pot deflection, i.e. 4 possible velocity settings in the forward or reverse motion directions.
Note. The scaling factor between encoder counts and mm/sec depends on the specific stage/actuator being driven.

SET:
Command structure (32 bytes)
6 byte header followed by 26 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
B0	04	1A	00	d|	s	Chan Ident		ZeroWnd		Vel1	

12	13	14	15	16	17	18	19	20	21	22	23
Data											
Vel1		Wnd1		Vel2				Wnd2		Vel3	

24	25	26	27	28	29	30	31
Data							
Vel3		Wnd3		Vel4			

Data Structure:

field	description	format
Chan Ident	The channel being addressed	word
ZeroWnd	The deflection from the mid position (in ADC counts 0 to 127) before motion can start	word
Vel1	The velocity (in encoder counts /sec) to move when between Wnd0 and PotDef1	long
Wnd1	The deflection from the mid position (in ADC counts, Wnd0 to 127) to apply Vel1	word
Vel2	The velocity (in encoder counts /sec) to move when between PotDef1 and PotDef2	long
Wnd2	The deflection from the mid position (in ADC counts, PotDef1 to 127) to apply Vel2	word

Vel3	The velocity (in encoder counts/sec) to move when between PotDef2 and PotDef3	long
Wnd3	The deflection from the mid position (in ADC counts PotDef2 to 127) to apply Vel3	word
Vel4	The velocity (in encoder counts /sec) to move when beyond PotDef3	long

Example: For the Z8 series motors, there are 512 encoder counts per revolution of the motor. The output shaft of the motor goes into a 67:1 planetary gear head. This requires the motor to rotate 67 times to rotate the 1.0 mm pitch lead screw one revolution. The end result is the lead screw advances by 1.0 mm.
Therefore, a 1 mm linear displacement of the actuator is given by

512 x 67 = 34,304 encoder counts

whereas the linear displacement of the lead screw per encoder count is given by

1.0 mm / 34,304 counts = 2.9 x 10-5 mm (29 nm).

Typical parameters settings Hex (decimal)
ZeroWnd – 14 (20)
Vel1 – 66, 0D,00,00 (3430)
Wnd1 – 32 (50)
Vel2 – CC, 1A, 00, 00 (6860)
Wnd2 – 50 (80)
Vel3 – 32, 28, 00, 00 (10290)
Wnd3 – 64 (100)
Vel4 – 00, 43, 00, 00 (17152)

Using the parameters above, no motion will start until the pot has been deflected to 20 (approx 1/6 full scale deflection), when the motor will start to move at 0.1mm/sec. At a deflection of 50 (approx 2/5 full scale deflection) the motor velocity will increase to 0.2mm/sec, and at 80, velocity will increase to 0.3 mm/sec. When the pot is deflected to 100 and beyond, the velocity will be 0.5 mm/sec.

Note. It is acceptable to set velocities equal to each other to reduce the number of speeds, however this is not allowed for the deflection settings, whereby the Wnd3 Pot Deflection value must be greater than Wnd2 Pot Deflection value.

TX *B0, 04, 1A, 00, D0, 01,* 01, 00, 01, 00, E8, 03, 00, 00, 00, 00, 00, 00, B0,35, 00, 00, CD, CC, CC, 00, 02, 00

Header: B0, 04, 1A, 00, D0, 01: Set Pot Params, 1AH (26) byte data packet, Generic USB Device.
Chan Ident: 01, 00: Channel 1 (always set to 1 for TDC001)
Wnd0: 14 (20 ADC Counts)
Vel1: 66, 0D,00,00 (3430 Encoder Counts/sec = 0.1 mm/sec)
PotDef1: 32 (50 ADC Counts)
Vel2: CC, 1A, 00, 00 (6860 Encoder Counts/sec = 0.2 mm/sec)
PotDef2: 50 (80 ADC Counts)

Vel3: 32, 28, 00, 00 (10290 Encoder Counts/sec = 0.3 mm/sec)
PotDef3: 64 (100 ADC Counts)
Vel4: 00, 43, 00, 00 (17152 Encoder Counts/sec = 0.5 mm/sec)

REQUEST:

Command structure (6 bytes):

0	1	2	3	4	5
header only					
17	04	Chan Ident	00	d	s

GET:

Response structure (28 bytes)
6 byte header followed by 22 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
B0	04	1A	00	d|	s	Chan Ident		ZeroWnd		Vel1	

12	13	14	15	16	17	18	19	20	21	22	23
Data											
Vel1		Wnd1		Vel2				Wnd2		Vel3	

24	25	26	27	28	29	30	31
Data							
Vel3		Wnd3		Vel4			

For structure see SET message above.

MGMSG_MOT_SET_BUTTONPARAMS 0x04B6
MGMSG_MOT_REQ_BUTTONPARAMS 0x04B7
MGMSG_MOT_GET_BUTTONPARAMS 0x04B8

Function: The control keypad can be used either to jog the motor, or to perform moves to absolute positions. This function is used to set the front panel button functionality.

SET:

Command structure (22 bytes)
6 byte header followed by 16 byte data packet as follows:

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td></tr>
<tr><td colspan="6"><i>header</i></td><td colspan="6"><i>Data</i></td></tr>
<tr><td>B6</td><td>04</td><td>0E</td><td>00</td><td>d|</td><td>s</td><td colspan="2">Chan Ident</td><td colspan="2">Mode</td><td colspan="2">Position1</td></tr>
</table>

<table>
<tr><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td></tr>
<tr><td colspan="10"><i>Data</i></td></tr>
<tr><td colspan="2">Position1</td><td colspan="4">Position2</td><td colspan="2">TimeOut</td><td colspan="2">Not Used</td></tr>
</table>

Data Structure:

field	description	format
Chan Ident	The channel being addressed	word
Mode	The buttons on the keypad can be used either to jog the motor (jog mode), or to perform moves to absolute positions (go to position mode). If set to 0x01, the buttons are used to jog the motor. Once set to this mode, the move parameters for the buttons are taken from the 'Jog' parameters set via the 'Move/Jogs' settings tab or the SetJogParams methods. If set to 0x02, each button can be programmed with a different position value (as set in the Position 1 and Position 2 parameters), such that the controller will move the motor to that position when the specific button is pressed.	word
Position1	The position (in encoder counts) to which the motor will move when the top button is pressed. This parameter is applicable only if 'Go to Position is selected in the 'Mode' parameter.	long
Position2	The position (in encoder counts) to which the motor will move when the bottom button is pressed. This parameter is applicable only if 'Go to Position is selected in the 'Mode' parameter.	long
TimeOut	A 'Home' move or can be performed by pressing and holding both buttons. Furthermore, the present position can be entered into the Position 1 or Position 2 parameter by holding down the associated button. The Time Out parameter specifies the time in ms that the button(s) must be depressed. This function is independent of the 'Mode' setting.	word
Not Used		word

Example: Set the button parameters for TDC001 as follows:
Mode: Go To Position
Position1: 0.5 mm
Position2: 1.2 mm
TimeOut: 2 secs

TX B6, 04, 10, 00, D0, 01, 01, 00, 02, 00, C0, 12, 00, 00, 00, 00, 00, 00, 00, 00

Header: B6, 04, 10, 00, D0, 01: SetButtonParams, 10H (16) byte data packet, Generic USB Device
Chan Ident: 01, 00: Channel 1 (always set to 1 for TDC001)
Mode: 02, 00 (i.e. Go to position)
Position1: 00, 43, 00, 00 (17152 Encoder Counts = 0.5 mm)
Position2: CC, A0, 00, 00 (41164 encoder counts = 1.2 mm):
TimeOut: D0, 07: (2 seconds)

REQUEST:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
DB	04	Chan Ident	00	d	s

GET:
Response structure (20 bytes)
6 byte header followed by 16 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
B6	04	0E	00	d|	s	Chan Ident		Mode		Position1	

12	13	14	15	16	17	18	19	20	21
Data									
Position1		Position2				TimeOut		Not Used	

For structure see SET message above.

MGMSG_MOT_SET_EEPROMPARAMS 0x04B9

Function: Used to save the parameter settings for the specified message. These settings may have been altered either through the various method calls or through user interaction with the GUI (specifically, by clicking on the 'Settings' button found in the lower right hand corner of the user interface).

SET:
Command structure (10 bytes)
6 byte header followed by 4 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9
header						*Data*			
B9	04	04	00	d|	s	Chan Ident		MsgID	

Data Structure:

field	description	format
Chan Ident	The channel being addressed	word
MsgID	The message ID of the message containing the parameters to be saved.	word

Example:

TX B9, 04, 04, 00, D0, 01, 01, 00, B6, 04,

Header: B9, 04, 04, 00, D0, 01: Set_EEPROMPARAMS, 04 byte data packet, Generic USB Device.
Chan Ident: 01, 00: Channel 1
MsgID: Save parameters specified by message 04B6 (SetButtonParams).

MGMSG_MOT_SET_PMDPOSITIONLOOPPARAMS 0x04D7
MGMSG_MOT_REQ_PMDPOSITIONLOOPPARAMS 0x04D8
MGMSG_MOT_GET_PMDPOSITIONLOOPPARAMS 0x04D9

Function: Used to set the position control loop parameters for the specified motor channel.

The motion processors within the BBD series controllers use a position control loop to determine the motor command output. The purpose of the position loop is to match the actual motor position and the demanded position. This is achieved by comparing the demanded position with the actual encoder position to create a position error, which is then passed through a digital PID-type filter. The filtered value is the motor command output.

SET:

Command structure (34 bytes)

6 byte header followed by 28 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
D7	04	1C	00	d|	s	Chan Ident		Kp Pos		Integral	

12	13	14	15	16	17	18	19	20	21	22	23
Data											
ILinPos				Differential		KdTimePos		KoutPos		KvffPos	

24	25	26	27	28	29	30	31	32	33
Data									
KaffPos		PosErrLim				N/A		N/A	

Data Structure:

field	description	format
Chan Ident	The channel being addressed	word
Proportional	The proportional gain. Together with the Integral and Differential, these terms determine the system response characteristics and accept values in the range 0 to 32767.	word
Integral	The integral gain. Together with the Proportional and Differential, these terms determine the system response characteristics and accept values in the range 0 to 32767.	word
Integral Limit	The Integral Limit parameter is used to cap the value of the Integrator to prevent runaway of the integral sum at the output. It accepts values in the range 0 to 7FFFFFFF. If set to 0 then the integration term in the PID loop is ignored.	dword
Differential	The differential gain. Together with the Proportional and Integral, these terms determine the system response characteristics and accept values in the range 0 to 32767.	word
KdTimePos	Under normal circumstances, the derivative term of the PID loop is recalculated at every servo cycle. However, it may be desirable to reduce the sampling rate to a lower value, in order to increase stability or simplify tuning. The KdTimePos parameter is used to set the sampling rate. For example, if	word

	set to 10, the derivative term is calculated every 10 servo cycles. The value is set in cycles, in the range 1 to 32767.	
KoutPos	The KoutPos parameter is a scaling factor applied to the output of the PID loop. It accepts values in the range 0 to 65535, where 0 is 0% and 65535 is 100%.	word
KvffPos	The KvffPos and KaffPos parameters are velocity and acceleration feed-forward terms that are added to the output of the PID filter to assist in tuning the motor drive signal. They accept values in the range 0 to 32767.	word
KaffPos		word
PosErrLim	Under certain circumstances, the actual encoder position may differ from the demanded position by an excessive amount. Such a large position error is often indicative of a potentially dangerous condition such as motor failure, encoder failure or excessive mechanical friction. To warn of, and guard against this condition, a maximum position error can be set in the PosErrLim parameter, in the range 0 to 7FFFFFFF. The actual position error is continuously compared against the limit entered, and if exceeded, the Motion Error bit (bit 15) of the Status Register is set and the associated axis is stopped.	dword
Not Used		word
Not Used		word

Example: Set the PID parameters for chan 2 as follows:
Proportional: 65
Integral: 175
Integral Limit: 80,000
Differential: 600
KdTimePos: 5
KoutPos: 5%
KvffPos: 0
KaffPos: 1000
PosErrLim: 65535

TX D7, 04, 1C, 00, A2, 01, 01, 00, 41, 00, AF, 00, 80, 38, 01, 00, 58, 02, 05, 00, CD, 0C, 00, 00, E8, 03, FF, FF, 00, 00, 00, 00

Header: D7, 04, 1C, 00, A2, 01: Set_PMDPositionLoopParams, 28 byte data packet, Channel 2.
Chan Ident: 01, 00: Channel 1 (always set to 1 for TDC001)
Proportional: *41, 00,:* Set the proportional term to 65
Integral: *AF, 00,:* Set the integral term to 175
Integral Limit: 80, 38, 01, 00,: Set the integral limit to 80,000
Differential: 58, 02,: Set the differential term to 600
KdTimePos: 05, 00,: Set the sampling rate to 5 cycles
KoutPos: CD, 0C,: Set the output scaling factor to 5% (i.e. 3277)
KvffPos: 00, 00,: Set the velocity feed forward value to zero
KaffPos: E8, 03,: Set the acceleration feed forward value to 1000
PosErrLim: FF, FF, 00, 00,: Set the position error limit to 65535.

REQUEST:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
D8	04	Chan Ident	00	d	s

GET:
Response structure (34 bytes)
6 byte header followed by 28 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
D9	04	1C	00	d|	s	Chan Ident		Kp Pos		Integral	

12	13	14	15	16	17	18	19	20	21	22	23
Data											
ILinPos				Differential		KdTimePos		KoutPos		KvffPos	

24	25	26	27	28	29	30	31	32	33
Data									
KaffPos		PosErrLim				N/A		N/A	

For structure see SET message above.

MGMSG_MOT_SET_PMDMOTOROUTPUTPARAMS 0x04DA
MGMSG_MOT_REQ_PMDMOTOROUTPUTPARAMS 0x04DB
MGMSG_MOT_GET_PMDMOTOROUTPUTPARAMS 0x04DC

Function: Used to set certain limits that can be applied to the motor drive signal. The individual limits are described below.

SET:

Command structure (20 bytes)

6 byte header followed by 14 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
DA	04	0E	00	d|	s	Chan Ident		Cont Current Lim		Energy Limit	

12	13	14	15	16	17	18	19
Data							
Motor Limit		Motor Bias		Not Used		Not Used	

Data Structure:

field	description	format
Chan Ident	The channel being addressed	word
ContCurrentLim	The system incorporates a current 'foldback' facility, whereby the continuous current level can be capped. The continuous current limit is set in the ContCurrentLim parameter, which accepts values as a percentage of maximum peak current, in the range 0 to 32767 (0 to 100%), which is the default maximum level set at the factory (this maximum value cannot be altered).	word
EnergyLim	When the current output of the drive exceeds the limit set in the ContCurrentLim parameter, accumulation of the excess current energy begins. The EnergyLim parameter specifies a limit for this accumulated energy, as a percentage of the factory set default maximum, in the range 0 to 32767 (0 to 100%). When the accumulated energy exceeds the value specified in the EnergyLim parameter, a 'current foldback' condition is said to exist, and the commanded current is limited to the value specified in the ContCurrentLim parameter. When this occurs, the Current Foldback status bit (bit 25) is set in the Status Register. When the accumulated energy above the ContCurrentLim value falls to 0, the limit is removed and the status bit is cleared.	word
MotorLim	The MotorLim parameter sets a limit for the motor drive signal and accepts values in the range 0 to 32767 (100%). If the system produces a value greater than the limit set, the motor command takes the limiting value. For example, if MotorLim is set to 30000 (91.6%), then signals greater than 30000 will be output as 30000 and values less than -30000 will be output as -30000.	word
MotorBias	When an axis is subject to a constant external force in one	word

	direction (such as a vertical axis pulled downwards by gravity) the servo filter can compensate by adding a constant DC bias to the output. This bias is set in the MotorBias parameter, which accepts values in the range -32767 to 32768. The default value is 0. Once set, the motor bias is applied while the position loop is enabled.	
Not Used		word
Not Used		word

Example: Set the motor output parameters for chan 2 as follows:
Continuous Current: 20%
Energy Limit: 14%
Motor Limit: 100%
Motor Bias: zero

TX DA, 04, 0E, 00, A2, 01, 01, 00, 99, 19, C0, 12, 00, 00, 00, 00, 00, 00, 00, 00

Header: DA, 04, 0E, 00, A2, 01: Set MotorOutputParams, 0EH (14) byte data packet, Channel 2.
Chan Ident: 01, 00: Channel 1 (always set to 1 for TDC001)
Cont Current Limit:
Energy Limit: 99, 19: Set the energy limit to 14%
Motor Limit: C0, 12: Set the motor limit to 100%
Motor Bias: 00, 00: Set the motor bias to zero

REQUEST:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
DB	04	Chan Ident	00	d	s

GET:
Response structure (20 bytes)
6 byte header followed by 14 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
DC	04	0E	00	d|	s	Chan Ident		Cont Current Lim		Energy Limit	

12	13	14	15	16	17	18	19
Data							
Motor Limit		Motor Bias		Not Used		Not Used	

For structure see SET message above.

MGMSG_MOT_SET_PMDTRACKSETTLEPARAMS 0x04E0
MGMSG_MOT_REQ_PMDTRACKSETTLEPARAMS 0x04E1
MGMSG_MOT_GET_PMDTRACKSETTLEPARAMS 0x04E2

Function: Moves are generated by an internal profile generator, and are based on either a trapezoidal or S-curve trajectory. A move is considered complete when the profile generator has completed the calculated move and the axis has 'settled' at the demanded position. This command contains parameters which specify when the system is settled.

Further Information

The system incorporates a monitoring function, which continuously indicates whether or not the axis has ‘settled’. The ‘Settled’ indicator is bit 14 in the Status Register and is set when the associated axis is settled. Note that the status bit is controlled by the processor, and cannot be set or cleared manually.

The axis is considered to be ‘settled’ when the following conditions are met:

* the axis is at rest (i.e. not performing a move),

* the error between the demanded position and the actual motor position is less than or equal to a specified number of encoder counts (0 to 65535) set in the *SettleWnd* parameter (Settle Window),

* the above two conditions have been met for a specified number of cycles (settle time, 1 cycle = 102.4 μs), set in the *SettleTime* parameter (range 0 to 32767).

The above settings are particularly important when performing a sequence of moves. If the PID parameters are set such that the settle window cannot be reached, the first move in the sequence will never complete, and the sequence will stall. The settle window and settle time values should be specified carefully, based on the required positional accuracy of the application. If positional accuracy is not a major concern, the settle time should be set to '0'. In this case, a move will complete when the motion calculated by the profile generator is completed, irrespective of the actual position attained, and the settle parameters described above will be ignored.

The processor also provides a ‘tracking window’, which is used to monitor servo performance outside the context of motion error. The tracking window is a programmable position error limit within which the axis must remain, but unlike the position error limit set in the SetDCPositionLoopParams method, the axis is not stopped if it moves outside the specified tracking window. This function is useful for processes that rely on the motor’s correct tracking of a set trajectory within a specific range. The tracking window may also be used as an early warning for performance problems that do not yet qualify as motion error.

The size of the tracking window (i.e. the maximum allowable position error while remaining within the tracking window) is specified in the *TrackWnd* parameter, in the range 0 to 65535. If the position error of the axis exceeds this value, the Tracking Indicator status bit (bit 13) is set to 0 in the Status Register. When the position error returns to within the window boundary, the status bit is set to 1.

SET:
Command structure (18 bytes)
6 byte header followed by 12 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
E0	04	0C	00	d|	s	Chan Ident		Time		Settle Window	

12	13	14	15	16	17
Data					
Track Window		Not Used		Not Used	

Data Structure:

field	description	format
Chan Ident	The channel being addressed	word
Time	The time that the associated axis must be settled before the 'Settled' status bit is set. The time is set in cycles, in the range 0 to 32767, 1 cycle = 102.4 µs.	word
Settle Window	The position error is defined as the error between the demanded position and the actual motor position. This parameter specifies the number of encoder counts (in the range 0 to 65535) that the position error must be less than or equal to, before the axis is considered 'settled'.	word
Track Window	The maximum allowable position error (in the range 0 to 65535) whilst tracking .	word
Not Used		word
Not Used		word

Example: Set the track and settle parameters for chan 2 as follows:
Settle Time: 20%
Settle Window: 14%
Track Window: 100%

s
TX E0, 04, 0C, 00, A2, 01, 01, 00, 00, 00, 14, 00, 00, 00, 00, 00, 00, 00, 00, 00

Header: E0, 04, 0C, 00, A2, 01: Set MotorOutputParams, 0CH (12) byte data packet, Channel 2.
Chan Ident: 01, 00: Channel 1 (always set to 1 for TDC001)
Time: 00, 00: Set the Settle time to zero
Settle Window: 14, 00: Set the settle window to 20 encoder counts
Track Window: 00, 00: Set the track window to zero

REQUEST:

Command structure (6 bytes):

0	1	2	3	4	5
header only					
E1	04	Chan Ident	00	d	s

GET:

Response structure (18 bytes)

6 byte header followed by 12 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
E2	04	0C	00	d|	s	Chan Ident		Time		Settle Window	

12	13	14	15	16	17
Data					
Track Window		Not Used		Not Used	

For structure see SET message above.

MGMSG_MOT_SET_PMDPROFILEMODEPARAMS 0x04E3
MGMSG_MOT_REQ_PMDPROFILEMODEPARAMS 0x04E4
MGMSG_MOT_GET_PMDPROFILEMODEPARAMS 0x04E5

Function: The system incorporates a trajectory generator, which performs calculations to determine the instantaneous position, velocity and acceleration of each axis at any given moment. During a motion profile, these values will change continuously. Once the move is complete, these parameters will then remain unchanged until the next move begins.

The specific move profile created by the system depends on several factors, such as the profile mode and profile parameters presently selected, and other conditions such as whether a motion stop has been requested. This method is used to set the profile mode to either 'Trapezoidal' or 'S-curve'.

SET:

Command structure (18 bytes)

6 byte header followed by 12 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
E3	04	0C	00	d|	s	Chan Ident		Mode		Jerk	

12	13	14	15	16	17
Data					
Jerk		Not Used		Not Used	

Data Structure:

field	description	format
Chan Ident	The channel being addressed	word
Mode	The move profile to be used: Trapezoidal: 0 S-Curve: 2 The Trapezoidal profile is a standard, symmetrical acceleration/deceleration motion curve, in which the start velocity is always zero. The S-curve profile is a trapezoidal curve with an additional 'Jerk' parameter, which limits the rate of change of acceleration and smooths out the contours of the motion profile. In this profile mode, the acceleration increases gradually from 0 to the specified acceleration value, then decreases at the same rate until it reaches 0 again at the specified velocity. The same sequence in reverse brings the axis to a stop at the programmed destination position.	word
Jerk	The Jerk value is specified in mm/s^3 in the Jerk parameter, and accepts values in the range 0 to 4294967295. It is used to specify the maximum rate of change in acceleration in a single cycle of the basic trapezoidal curve. 1.0 mm/s^3 is equal to 92.2337 jerk units.	dword
Not Used		word
Not Used		word

Example: Set the profile mode parameters for chan 2 as follows:
Profile Mode: S-curve
Jerk: 10,000 mm^3

TX E3, 04, 0C, 00, A2, 01, 01, 00, 02, 00, E1, 12, 0E, 00, 00, 00, 00, 00,

Header: E3, 04, 0C, 00, A2, 01: Set ProfileModeParams, 0CH (12) byte data packet, Channel 2.
Chan Ident: 01, 00: Channel 1 (always set to 1 for TDC001)
Profile Mode: 02, 00: Set the profile mode to S-Curve
Jerk: E1, 12,0E, 00: Set the jerk value to 10,000 mm/sec^3 (i.e. 922337)

REQUEST:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
E4	04	Chan Ident	00	d	s

GET:
Response structure (18 bytes)
6 byte header followed by 12 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
E5	04	0C	00	d|	s	Chan Ident		Mode		Jerk	

12	13	14	15	16	17
Data					
Jerk		Not Used		Not Used	

For structure see SET message above.

MGMSG_MOT_SET_PMDCURRENTLOOPPARAMS 0x04D4
MGMSG_MOT_REQ_PMDCURRENTLOOPPARAMS 0x04D5
MGMSG_MOT_GET_PMDCURRENTLOOPPARAMS 0x04D6

Function: Used to set the current control loop parameters for the specified motor channel.

The motion processors within the BBD series controllers use digital current control as a technique to control the current through each phase winding of the motors. In this way, response times are improved and motor efficiency is increased. This is achieved by comparing the required (demanded) current with the actual current to create a current error, which is then passed through a digital PI-type filter. The filtered current value is used to develop an output voltage for each motor coil.

This method sets various constants and limits for the current feedback loop.

SET:

Command structure (24 bytes)

6 byte header followed by 18 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
D4	04	12	00	d|	s	Chan Ident		Phase		KpCurrent	

12	13	14	15	16	17	18	19	20	21	22	23
Data											
KiCurrent		ILimCurrent		DeadBand		Kff		Not Used		Not Used	

Data Structure:

field	description	format
Chan Ident	The channel being addressed	word
Phase	The current phase to set: PHASEA 0 PHASEB 1 PHASEA AND B 2	word
KpCurrent	The proportional gain. Together with the KiCurrent this term determines the system response characteristics and accept values in the range 0 to 32767.	word
KiCurrent	The integral gain. Together with the KpCurrent this term determines the system response characteristics and accept values in the range 0 to 32767.	word
ILimCurrent	The ILimCurrent parameter is used to cap the value of the Integrator to prevent runaway of the integral sum at the output. It accepts values in the range 0 to 32767. If set to 0 then the integration term in the PID loop is ignored.	word
IDeadBand	The IDeadBand parameter allows an integral dead band to be set, such that when the error is within this dead band, the integral action stops, and the move is completed using the proportional term only. It accepts values in the range 0	word

	to 32767.	
Kff	The Kff parameter is a feed-forward term that is added to the output of the PID filter to assist in tuning the motor drive signal. It accepts values in the range 0 to 32767.	word
Not Used		word
Not Used		word

Example: Set the limit switch parameters for chan 2 as follows:
Phase: A and B
KpCurrent: 35
KiCurrent: 80
ILimCurrent: 32,767
DeadBand: 50
Kff: 0

TX D4, 04, 12, 00, A2, 01, 01, 00, 02, 00, 23, 00, 50, 00, FF, 7F, 32, 00, 00, 00, 00, 00, 00, 00,

Header: D4, 04, 12, 00, A2, 01: Set_PMDCurrentLoopParams, 18 byte data packet, Channel 2.
Chan Ident: 01, 00: Channel 1 (always set to 1 for TDC001)
Phase: 02, 00: Set Phase A and Phase B
KpCurrent: *23, 00,:* Set the proportional term to 35
KiCurrent: *50, 00,:* Set the integral term to 80
ILimCurrent: FF, 7F,: Set the integral limit to 32767
IDeadBand: 32, 00,: Set the deadband to 50
Kff: 00, 00: Set the feed forward value to zero

REQUEST:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
D8	04	Chan Ident	00	d	s

GET:
Command structure (24 bytes)
6 byte header followed by 18 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
D6	04	12	00	d|	s	Chan Ident		Phase		KpCurrent	

12	13	14	15	16	17	18	19	20	21	22	23
Data											
KiCurrent		ILimCurrent		DeadBand		Kff		Not Used		Not Used	

For structure see SET message above.

MGMSG_MOT_SET_PMDSETTLEDCURRENTLOOPPARAMS 0x04E9
MGMSG_MOT_REQ_PMDSETTLEDCURRENTLOOPPARAMS 0x04EA
MGMSG_MOT_GET_PMDSETTLEDCURRENTLOOPPARAMS 0x04EB

Function: These commands assist in maintaining stable operation and reducing noise at the demanded position. They allow the system to be tuned such that errors caused by external vibration and manual handling (e.g. loading of samples) are minimized, and are applicable only when the stage is settled, i.e. the Axis Settled status bit (bit 14) is set.

SET:

Command structure (24 bytes)

6 byte header followed by 18 byte data packet as follows:

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td></tr>
<tr><td colspan="6"><i>header</i></td><td colspan="6"><i>Data</i></td></tr>
<tr><td>E9</td><td>04</td><td>12</td><td>00</td><td>d|</td><td>s</td><td colspan="2">Chan Ident</td><td colspan="2">Phase</td><td colspan="2">KpSettled</td></tr>
</table>

<table>
<tr><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td></tr>
<tr><td colspan="12"><i>Data</i></td></tr>
<tr><td colspan="2">KiSettled</td><td colspan="2">ILimSettled</td><td colspan="2">DeadBandSet</td><td colspan="2">KffSettled</td><td colspan="2">Not Used</td><td colspan="2">Not Used</td></tr>
</table>

Data Structure:

field	**description**	**format**
Chan Ident	The channel being addressed	word
Phase	The current phase to set: PHASEA 0 PHASEB 1 PHASEA AND B 2	word
KpSettled	The proportional gain. Together with the KiSettled this term determines the system response characteristics and accept values in the range 0 to 32767.	word
KiSettled	The integral gain. Together with the KpSettled this term determines the system response characteristics and accept values in the range 0 to 32767.	word
ILimSettled	The ILimSettled parameter is used to cap the value of the Integrator to prevent runaway of the integral sum at the output. It accepts values in the range 0 to 32767. If set to 0 then the integration term in the PID loop is ignored.	word
IDeadBandSettled	The IDeadBandSettled parameter allows an integral dead band to be set, such that when the error is within this dead band, the integral action stops, and the move is completed using the proportional term only. It accepts values in the range 0 to 32767.	word
KffSettled	The KffSettled parameter is a feed-forward term that is added to the output of the PID filter to assist in tuning the motor drive signal. It accepts values in the range 0 to 32767.	word
Not Used		word
Not Used		word

Example: Set the limit switch parameters for chan 2 as follows:
Phase: A and B
KpSettled: 0
KiSettled: 40
ILimSettled: 30,000
DeadBandSettled: 50
KffSettled:500

TX E9, 04, 12, 00, A2, 01, 01, 00, 02, 00, 00, 00, 28, 00, 30, 75, 32, 00, F4, 01, 00, 00, 00, 00,

Header: D4, 04, 12, 00, A2, 01: Set_PMDSettledCurrentLoopParams, 18 byte data packet, Channel 2.
Chan Ident: 01, 00: Channel 1 (always set to 1 for TDC001)
Phase: 02, 00: Set Phase A and Phase B
KpCurrent: *00, 00,:* Set the proportional term to zero
KiCurrent: *28, 00,:* Set the integral term to 40
ILimCurrent: 30, 75,: Set the integral limit to 30,000
IDeadBand: 32, 00,: Set the deadband to 50
Kff: F4, 01: Set the feed forward value to 500

REQUEST:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
D8	04	Chan Ident	00	d	s

GET:
Command structure (24 bytes)
6 byte header followed by 18 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
EB	04	12	00	d|	s	Chan Ident		Phase		KpSettled	

12	13	14	15	16	17	18	19	20	21	22	23
Data											
KiSettled		ILimSettled		DeadBandSet		KffSettled		Not Used		Not Used	

For structure see SET message above.

MGMSG_MOT_SET_PMDSTAGEAXISPARAMS 0x04F0
MGMSG_MOT_REQ_PMDSTAGEAXISPARAMS 0x04F1
MGMSG_MOT_GET_PMDSTAGEAXISPARAMS 0x04F2

Function: The REQ and GET commands are used to obtain various parameters pertaining to the particular stage being driven. Most of these parameters are inherent in the design of the stage and cannot be altered. The SET command can only be used to increase the Minimum position value and decrease the Maximum position value, thereby reducing the overall travel of the stage.

SET:

Command structure (80 bytes)

6 byte header followed by 74 byte data packet – see Get for structure

REQUEST:

Command structure (6 bytes):

0	1	2	3	4	5
header only					
F1	04	Chan Ident	00	d	s

GET:

Command structure (80 bytes)

6 byte header followed by 74 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
F2	04	12	00	d|	s	Chan ID		Stage ID		Axis ID	

12	13	14	15	16	17	18	19	20	21	22	23
Data											
Part No/Axis											

24	25	26	27	28	29	30	31	32	33	34	35
Data											
Part No/Axis				Serial Number				Counts per Unit			

36	37	38	39	40	41	42	43	44	45	46	47
Data											
MinPos				Max Pos				Max Accn			

48	49	50	51	52	53	54	55	56	57	58	59
Data											
Max Dec				Max Vel				Reserved		Reserved	

60	61	62	63	64	65	66	67	68	69	70	71
Data											
Reserved		Reserved		Reserved				Reserved			

72	73	74	75	76	77	78	79
Data							
Reserved				Reserved			

Data Structure:

field	description	format
Stage ID	This 2 byte parameter identifies the stage and axis: 00, 10 - MLS203_X_AXIS 00, 11 - MLS203_Y_AXIS	word
AxisID	Not used for the BBD series controllers	word
PartNoAxis	A 16 byte character string used to identify the stage type and axis being driven.	char
SerialNum	The Serial number of the stage	dword
CntsPerUnit	The number of encoder counts per real world unit (either mm or degrees).	dword
MinPos	The minimum position of the stage, typically zero	long
MaxPos	The maximum position of the stage in encoder counts	long
MaxAccn	The maximum acceleration of the stage in encoder counts per cycle per cycle	long
MaxDec	The maximum deceleration of the stage in encoder counts per cycle per cycle	long
MaxVel	The maximum velocity of the stage in encoder counts per cycle.	long
Reserved		word
Reserved		word
Reserved		word
Reserved		word
Reserved		dword
Reserved		dword
Reserved		dword
Reserved		dword

Example: Get the stage and axis parameters for chan 2:

TX F2, 04, 4A, 00, A2, 01, 01, 00, 11, 00, 00, 00, 4D,4C, 53, 32, 30, 33, 20, 59, 20, 41, 78, 69, 73, 00, 00, 00, 81, 96, 98, 00, 20, 4E, 00, 00, 00, 00, 00, 00, 60, E3, 16, 00, 60, 6B, 00, 00, 60, 6B, 00, 00, 9A, 99, 99, 01, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00

Header: F2, 04, 4A, 00, A2, 01: Get_PMDStageAxisParams, 74 byte data packet, Channel 2.
Chan Ident: 01, 00: Channel 1 (always set to 1 for TDC001)
Stage ID: 11, 00: MLS203 Y Axis
Axis ID: *00, 00,:* Not used
PartNo Axis: *4D, 4C, 53, 32, 30, 33, 20, 59, 20, 41, 78, 69, 73, 00, 00, 00,:*
M L S 2 0 3 Y A X I S
SerialNum: 81, 96, 98, 00
CntsPerUnit 20, 4E, 00, 00: the encoder counts per unit is set to 20000
MinPos: 00, 00, 00, 00: the feed minimum position is set to zero
MaxPos: 60, E3, 16, 00: the maximum position is set to 1500000
MaxAccn: 60, 6B, 00, 00: the maximum acceleration is set to 27488
MaxDec: 60, 6B, 00, 00: the maximum deceleration is set to 27488
MaxVel: 9A, 99, 99, 01: the maximum velocity is set to 26843546

MGMSG_MOT_GET_STATUSUPDATE 0x0481

Function: This message is returned when a status update is requested for the specified motor channel. This request can be used instead of enabling regular updates as described above.

GET:

Status update messages are received with the following format:-

Response structure (20 bytes)

6 byte header followed by 14 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
81	04	0E	00	d|	s	Chan Ident		Position			

12	13	14	15	16	17	18	19
Data							
EncCount				Status Bits			

Data Structure:

field	description	format
Chan Ident	The channel being addressed is always P_MOD_CHAN1 (0x01) encoded as a 16-bit word (0x01 0x00)	word
Position	The position encoder count. In the APT Stepper Motor controllers the encoder resolution is 25,600 counts per mm, therefore a position change of 1 mm would be seen as this parameter changing by 25,600. The LONG variable is a 32 bit value, encoded in the data stream in the Intel format.	long
EncCount	For use with encoded stages only.	long
Status Bits	The meaning of individual bits in this 32-bit variable is described in the bit mask table below (1 = active, 0 = inactive).	dword

bit mask	meaning
0x00000001	forward (CW) hardware limit switch is active
0x00000002	reverse (CCW) hardware limit switch is active
0x00000004	forward (CW) software limit switch is active
0x00000008	reverse (CCW) software limit switch is active
0x00000010	in motion, moving forward (CW)
0x00000020	in motion, moving reverse (CCW)
0x00000040	in motion, jogging forward (CW)
0x00000080	in motion, jogging reverse (CCW)
0x00000100	motor connected
0x00000200	in motion, homing
0x00000400	homed (homing has been completed)
0x00001000	interlock state (1 = enabled)

This is not full list of all the bits but the remaining bits reflect information about the state of the hardware that in most cases does not affect motion.

MGMSG_MOT_REQ_STATUSUPDATE 0x0480

Function: Used to request a status update for the specified motor channel. This request can be used instead of enabling regular updates as described above.

REQUEST:

Command structure (6 bytes):

0	1	2	3	4	5
header only					
90	04	Chan Ident	00	d	s

GET:

See previous details on MGMSG_MOT_GET_STATUSUPDATE 0x0481.

MGMSG_MOT_GET_DCSTATUSUPDATE 0x0491

Function: This message is returned when a status update is requested for the specified motor channel. This request can be used instead of enabling regular updates as described above.

GET:
Status update messages are received with the following format:-

Response structure (20 bytes)
6 byte header followed by 14 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
91	04	0E	00	d|	s	Chan Ident		Position			

12	13	14	15	16	17	18	19
Data							
Velocity		Reserved		Status Bits			

Data Structure:

field	description	format
Chan Ident	The channel being addressed is always P_MOD_CHAN1 (0x01) encoded as a 16-bit word (0x01 0x00)	word
Position	The position encoder count. In the BBD10X series controllers the encoder resolution is 20,000 counts per mm, therefore a position change of 1 mm would be seen as this parameter changing by 20,000 (twenty thousand). The LONG variable is a 32 bit value, encoded in the data stream in the Intel format, so for example a position of 1 million encoder counts (equivalent to 50 mm) would be sent as byte stream 0x40, 0x42, 0x0F, 0x00 since 1 million is hexadecimal 0xF4240.	long
Velocity	The actual velocity. Scaling is 204.8 per mm/sec, so a real-life measured speed of 100 mm/sec is read as 205. Again, the two-byte data stream will be encoded in the Intel format.	word
Reserved	Currently Not Used	Word
Status Bits	The meaning of individual bits in this 32-bit variable is described in the bit mask table below	dword

bit mask	meaning
0x00000001	forward hardware limit switch is active
0x00000002	reverse hardware limit switch is active
0x00000010	in motion, moving forward
0x00000020	in motion, moving reverse
0x00000040	in motion, jogging forward
0x00000080	in motion, jogging reverse
0x00000200	in motion, homing

0x00000400	homed (homing has been completed)
0x00001000	tracking
0x00002000	settled
0x00004000	motion error (excessive position error)
0x01000000	motor current limit reached
0x80000000	channel is enabled

This is not full list of all the bits but the remaining bits reflect information about the state of the hardware that in most cases does not affect motion.

MGMSG_MOT_REQ_DCSTATUSUPDATE 0x0490

Function: Used to request a status update for the specified motor channel. This request can be used instead of enabling regular updates as described above.

REQUEST:

Command structure (6 bytes):

0	1	2	3	4	5
header only					
90	04	Chan Ident	00	d	s

GET:

See previous details on MGMSG_MOT_GET_DCSTATUSUPDATE 0x0491.

MGMSG_MOT_ACK_DCSTATUSUPDATE 0x0492

Only Applicable If Using USB COMMS. Does not apply to RS-232 COMMS

Function: If using the USB port, this message called “server alive” must be sent by the server to the controller at least once a second or the controller will stop responding after ~50 commands.
The controller keeps track of the number of "status update" type of messages (e.g.move complete message) and it if has sent 50 of these without the server sending a "server alive" message, it will stop sending any more "status update" messages.
This function is used by the controller to check that the PC/Server has not crashed or switched off. There is no response.

Structure (6 bytes):

0	1	2	3	4	5
header only					
62	06	00	00	d	s

TX 92, 04, 00, 00, 21, 01

MGMSG_ MOT_REQ_STATUSBITS 0x0429
MGMSG_ MOT_GET_STATUSBITS 0x042A

Function: Used to request a "cut down" version of the status update message, only containing the status bits, without data about position and velocity.

SET: N/A

REQUEST:

Command structure (6 bytes):

0	1	2	3	4	5
header only					
29	04	Chan Ident	00	d	s

GET:

Response structure (12 bytes)

6 byte header followed by 6 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
2A	04	06	00	d|	s	Chan Ident		Status Bits			

Data Structure:

field	description	format
Chan Ident	The channel being addressed	Word
Status Bits	The status bits are assigned exactly as described in the section detailing the MGMSG_MOT_GET_DCSTATUSUPDATE command.	DWord

MGMSG_ MOT_SUSPEND_ENDOFMOVEMSGS 0x046B

Function: Sent to disable all unsolicited end of move messages and error messages returned by the controller, i.e.

MGMSG_MOT_MOVE_STOPPED
MGMSG_MOT_MOVE_COMPLETED
MGMSG_MOT_MOVE_HOMED

The command also disables the error messages that the controller sends when an error conditions is detected:

MGMSG_HW_RESPONSE
MGMSG_HW_RICHRESPONSE

This is useful in single threaded custom client applications that are not configured to accept unsolicited messages from the controller at any time.

Command structure (6 bytes):

0	1	2	3	4	5
header only					
6B	04	00	00	d	s

MGMSG_ MOT_RESUME_ENDOFMOVEMSGS 0x046C

Function: Sent to resume all unsolicited end of move messages and error messages returned by the controller, i.e.

MGMSG_MOT_MOVE_STOPPED
MGMSG_MOT_MOVE_COMPLETED
MGMSG_MOT_MOVE_HOMED

The command also disables the error messages that the controller sends when an error conditions is detected:

MGMSG_HW_RESPONSE
MGMSG_HW_RICHRESPONSE

This is the default state when the controller is powered up.

Command structure (6 bytes):

0	1	2	3	4	5
header only					
6C	04	00	00	d	s

Solenoid Control Messages

Introduction

The APT Solenoid drive uses the Motor server control instance control its functionality. The messages listed here provide the extra functionality required for a client application to control one or more of the Thorlabs series of TSC001 T-Cube solenoid driver units.

MGMSG_MOT_SET_SOL_OPERATINGMODE 0x04C0
MGMSG_MOT_REQ_SOL_OPERATINGMODE 0x04C1
MGMSG_MOT_GET_SOL_OPERATINGMODE 0x04C2

Function: This message sets the operating mode of the solenoid driver.

SET:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
C0	04	Chan Ident	Mode	d	s

Data Structure:

field	description	format
Chan Ident	The channel being addressed	char
Operating Mode	The operating mode of the unit as a 4 bit integer: 0x01 SOLENOID_MANUAL - In this mode, operation of the solenoid is via the front panel 'Enable' button, or by the 'Output' buttons on the GUI panel. 0x02 SOLENOID_SINGLE - In this mode, the solenoid will open and close each time the front panel 'Enable' button is pressed, or the 'Output ON' button on the GUI panel is clicked. The ON and OFF times are specified by calling the MGMSG_MOT_SET_SOL_CYCLEPARAMS message. 0x03 SOLENOID_AUTO - In this mode, the solenoid will open and close continuously after the front panel 'Enable' button is pressed, or the 'Output ON' button on the GUI panel is clicked. The ON and OFF times, and the number of cycles performed, are specified by calling the MGMSG_MOT_SET_SOL_CYCLEPARAMS message. 0x04 SOLENOID_TRIGGER - In Triggered mode, a rising edge on rear panel TRIG IN BNC input will start execution of the parameters programmed on the unit (On Time, Off Time, Num Cycles - see MGMSG_MOT_SET_SOL_CYCLEPARAMS message.). The unit must be primed (i.e. the ENABLE button pressed and the ENABLED LED lit) before the unit can respond to the external trigger.	char

Example: Set the control mode to 'Single'.

TX C0, 04, 01, 02, 50, 01

C0,04 SET_SOL_OPERATINGMODE
01, Channel 1
02, Set mode to 'Single'
50, destination Generic USB device
01, Source PC

REQ:

Command structure (6 bytes):

0	1	2	3	4	5
header only					
C1	04	Chan Ident	00	d	s

Example: Request the control mode

TX C1, 04, 01, 00, 50, 01

GET:

Response structure (6 bytes):

0	1	2	3	4	5
header only					
C2	04	Chan Ident	Mode	d	s

Example: Get the control mode currently set.

RX C2, 04, 01, 01, 01, 50

MGMSG_MOT_SET_SOL_CYCLEPARAMS 0x04C3
MGMSG_MOT_REQ_SOL_CYCLEPARAMS 0x04C4
MGMSG_MOT_GET_SOL_CYCLEPARAMS 0x04C5

Function: Used to set the cycle parameters that are applicable when the solenoid controller is operating in one of the non-manual modes.

SET:

Command structure (20 bytes)

6 byte header followed by 14 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
C3	04	0E	00	d|	s	Chan Ident		OnTime			

12	13	14	15	16	17	18	19
Data							
OffTime				NumCycles			

Data Structure:

field	description	format
Chan Ident	The channel being addressed	word
OnTime	The time which the solenoid is activated (100ms to 10,000s in 250 µs steps)	long
OffTime	The time which the solenoid is a de-activated (100ms to 10,000s in 250 µs steps)	long
NumCycles	If the unit is operating in 'Auto' mode, the number of Open/Close cycles to perform. (0 to 1,000,000) is specified in the NumCycles parameter. If set to '0' the unit cycles indefinitely. If the unit is not operating in 'Auto' mode, the NumCycles parameter is ignored.	long

Example: Set the cycle parameters parameters for chan 1 as follows:
OnTime: 1000ms
OffTime: 1000ms
NumCycles: 20

TX C3, 04, 0E, 00, D0, 01, 01, 00, A0, 0F, 00, 00, A0, 0F, 00, 00, 14, 00, 00, 00

Header: C3, 04, 0E, 00, D0, 01: Set Cycle Params, D0H (14) byte data packet, Generic USB Device.
Chan Ident: 01, 00: Channel 1 (always set to 1 for TSC001)
OnTime: *A0, 0F, 00, 00:* Set on time to 1000 ms (i.e. 4000 x 250 µs)
OffTime: *A0, 0F, 00, 00:* Set off time to 1000 ms (i.e. 4000 x 250 µs)
NumCycles: *14, 00, 00, 00:* Set number of cycles to 20

REQUEST:

Command structure (6 bytes):

0	1	2	3	4	5
header only					
C4	04	Chan Ident	00	d	s

GET:

Response structure (20 bytes)

6 byte header followed by 14 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
C5	04	0E	00	d|	s	Chan Ident		OnTime			

12	13	14	15	16	17	18	19
Data							
OffTime				NumCycles			

For structure see SET message above.

MGMSG_MOT_SET_SOL_INTERLOCKMODE 0x04C6
MGMSG_MOT_REQ_SOL_INTERLOCKMODE 0x04C7
MGMSG_MOT_GET_SOL_INTERLOCKMODE 0x04C8

Function: The solenoid unit features a hardware interlock jackplug. This message specifies whether the solenoid driver requires the hardware interlock to be fitted before it can operate.

SET:

Command structure (6 bytes):

0	1	2	3	4	5
header only					
C6	04	Chan Ident	Mode	d	s

Data Structure:

field	description	format
Chan Ident	The channel being addressed	char
Interlock Mode	The operating mode of the unit as a 4 bit integer: 0x01 SOLENOID_ENABLED – The hardware interlock must be fitted before the unit can be operated. 0x02 SOLENOID_DISABLED – The hardware interlock is not required.	char

Example: Set the interlock mode to 'Enabled'.

TX C6, 04, 01, 01, 50, 01

C0,06 SET_SOL_INTERLOCKMODE
01, Channel 1
01, Set mode to 'Enabled'
50, destination Generic USB device
01, Source PC

REQ:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
C7	04	Chan Ident	00	d	s

Example: Request the control mode

TX C7, 04, 01, 00, 50, 01

GET:
Response structure (6 bytes):

0	1	2	3	4	5
header only					
C8	04	Chan Ident	Mode	d	s

Example: Get the control mode currently set.

RX C8, 04, 01, 01, 01, 50

MGMSG_MOT_SET_SOL_STATE 0x04CB
MGMSG_MOT_REQ_SOL_STATE 0x04CC
MGMSG_MOT_GET_SOL_STATE 0x04CD

Function: This message sets the output state of the solenoid unit, and overrides any existing settings.

SET:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
CB	04	Chan Ident	State	d	s

Data Structure:

field	description	format
Chan Ident	The channel being addressed	char
Interlock Mode	The operating mode of the unit as a 4 bit integer: 0x01 SOLENOID_ON – The solenoid is active. 0x02 SOLENOID_OFF – The solenoid is de-activated.	char

Example: Set the solenoid to 'ON'.

TX CB, 04, 01, 01, 50, 01

CB,06 SET_SOL_STATE
01, Channel 1
01, Set state to 'ON'
50, destination Generic USB device
01, Source PC

REQ:

Command structure (6 bytes):

0	1	2	3	4	5
header only					
CC	04	Chan Ident	00	d	s

Example: Request the control mode

TX CC, 04, 01, 00, 50, 01

GET:

Response structure (6 bytes):

0	1	2	3	4	5
header only					
CD	04	Chan Ident	Mode	d	s

Example: Get the control mode currently set.

RX CD, 04, 01, 01, 01, 50

Piezo Control Messages

Introduction

The ‘Piezo’ control messages provide the functionality required for a client application to control one or more of the Thorlabs series of piezo controller units. This range of controllers covers both open and closed loop piezo control in a variety of formats including compact Cube type controllers, benchtop units and 19” rack based modular drivers. **Note.** For ease of description, the TSG001 T-Cube Strain Gauge reader is considered here as a piezo controller. The list of controllers covered by the piezo messages includes:-

BPC001 – 1 Channel Benchtop Piezo Driver
BPC002 – 2 Channel Benchtop Piezo Driver
MPZ601 – 2 Channel Modular Piezo Driver
BPC101 – 1 Channel Benchtop Piezo Driver (2006 onwards)
BPC102 – 2 Channel Benchtop Piezo Driver (2006 onwards)
BPC103 – 3 Channel Benchtop Piezo Driver (2006 onwards)
BPC201 – 1 Channel Benchtop Piezo Driver (2007 onwards)
BPC202 – 2 Channel Benchtop Piezo Driver (2007 onwards)
BPC203 – 3 Channel Benchtop Piezo Driver (2007 onwards)
BPC301 – 1 Channel Benchtop Piezo Driver (2011 onwards)
TPZ001 – 1 Channel T-Cube Piezo Driver
TSG001 – 1 Channel T-Cube Strain Gauge Reader

The piezo messages can be used to perform activities such as selecting output voltages, reading the strain gauge position feedback, operating open and closed loop modes and enabling force sensing mode. With a few exceptions, these messages are generic and apply equally to both single and dual channel units.

Where applicable, the target channel is identified in the lChanID parameter and on single channel units, this must be set to CHAN1_ID. On dual channel units, this can be set to CHAN1_ID, CHAN2_ID or CHANBOTH_ID as required.

For details on the operation of the Piezo Controller, and information on the principles of operation, refer to the handbook supplied with the unit.

MGMSG_PZ_SET_POSCONTROLMODE 0x0640
MGMSG_PZ_REQ_POSCONTROLMODE 0x0641
MGMSG_PZ_GET_POSCONTROLMODE 0x0642

Function: When in closed-loop mode, position is maintained by a feedback signal from the piezo actuator. This is only possible when using actuators equipped with position sensing.

This method sets the control loop status The Control Mode is specified in the Mode parameter as follows:

0x01	Open Loop (no feedback)
0x02	Closed Loop (feedback employed)
0x03	Open Loop Smooth
0x04	Closed Loop Smooth

If set to Open Loop Smooth or Closed Loop Smooth is selected, the feedback status is the same as above however the transition from open to closed loop (or vise versa) is achieved over a longer period in order to minimize voltage transients (spikes).

SET:

Command structure (6 bytes):

0	1	2	3	4	5
header only					
40	06	Chan Ident	Mode	d	s

Example: Set the control mode to closed loop.

TX 40, 06, 01, 02, 50, 01

REQ:

Command structure (6 bytes):

0	1	2	3	4	5
header only					
41	06	Chan Ident	00	d	s

Example: Request the control mode

TX 41, 06, 01, 00, 50, 01

GET:

Response structure (6 bytes):

0	1	2	3	4	5
header only					
42	06	Chan Ident	Mode	d	s

Example: Get the control mode currently set.

RX 42, 06, 01, 02, 01, 50

MGMSG_PZ_SET_OUTPUTVOLTS 0x0643
MGMSG_PZ_REQ_OUTPUTVOLTS 0x0644
MGMSG_PZ_GET_OUTPUTVOLTS 0x0645

Function: Used to set the output voltage applied to the piezo actuator. This command is applicable only in Open Loop mode. If called when in Closed Loop mode it is ignored.

SET:

Command structure (10 bytes)
6 byte header followed by 4 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9
header						*Data*			
43	06	04	00	d|	s	Chan Ident		Voltage	

Data Structure:

field	description	format
Chan Ident	The channel being addressed	word
Voltage	The output voltage applied to the piezo when operating in open loop mode. The voltage is set in the range -32768 to 32767 (-7FFF to 7FFF) to which corresponds to -100% to 100% of the maximum output voltage as set using the TPZ_IOSETTINGS command.	short

Example: Set the drive voltage to 70V

TX 43, 06, 04, 00, D0, 01, 01, 00, 77, 77,

Header: 43, 06, 04, 00, D0, 01: SetPZOutputVolts, 04 byte data packet, Generic USB Device.
Chan Ident: 01, 00: Channel 1
Voltage: *77, 77:* corresponds to 70 V (30583) for a max 75 V unit

REQUEST:

Command structure (6 bytes):

0	1	2	3	4	5
header only					
44	6	Chan Ident	00	d	s

GET:

Response structure (10 bytes)
6 byte header followed by 4 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9
header						*Data*			
45	06	04	00	d|	s	Chan Ident		Voltage	

For structure see SET message above.

MGMSG_PZ_SET_OUTPUTPOS 0x0646
MGMSG_PZ_REQ_OUTPUTPOS 0x0647
MGMSG_PZ_GET_OUTPUTPOS 0x0648

Function: Used to set the output position of piezo actuator. This command is applicable only in Closed Loop mode. If called when in Open Loop mode it is ignored. The position of the actuator is relative to the datum set for the arrangement using the ZeroPosition method.

SET:
Command structure (10 bytes)
6 byte header followed by 4 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9
header						*Data*			
46	06	04	00	d|	s	Chan Ident		PositionSW	

Data Structure:

field	description	format
Chan Ident	The channel being addressed	word
PositionSW	The output position of the piezo relative to the zero position. The voltage is set in the range 0 to 32767 (0 to 7FFF) which corresponds to 0 to 100% of the maximum piezo extension.	word

Example: Set the drive position to 15 µm (when total travel = 100 µm).

TX 46, 06, 04, 00, D0, 01, 01, 00, 66, 26,

Header: 43, 06, 04, 00, D0, 01: SetPZOutputPos, 04 byte data packet, Generic USB Device.
Chan Ident: 01, 00: Channel 1
Voltage: *33, 13:* corresponds to 15 µm for a max 100 µm unit

REQUEST:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
47	06	Chan Ident	00	d	s

GET:
Response structure (10 bytes)
6 byte header followed by 4 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9
header						*Data*			
48	06	04	00	d|	s	Chan Ident		PositionSW	

For structure see SET message above.

MGMSG_PZ_SET_INPUTVOLTSSRC 0x0652
MGMSG_PZ_REQ_INPUTVOLTSSRC 0x0653
MGMSG_PZ_GET_INPUTVOLTSSRC 0x0654

Function: Used to set the input source(s) which controls the output from the HV amplifier circuit (i.e. the drive to the piezo actuators).

SET:

Command structure (10 bytes)
6 byte header followed by 4 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9
header						*Data*			
52	06	04	00	d|	s	Chan Ident		PositionSW	

Data Structure:

field	description	format
Chan Ident	The channel being addressed	word
VoltSrc	The following values are entered into the VoltSrc parameter to select the various analog sources. *0x00 Software Only*: Unit responds only to software inputs and the HV amp output is that set using the SetVoltOutput method or via the GUI panel. *0x01 External Signal*: Unit sums the differential signal on the rear panel EXT IN (+) and EXT IN (-)connectors with the voltage set using the SetVoltOutput method *0x02 Potentiometer*: The HV amp output is controlled by a potentiometer input (either on the control panel, or connected to the rear panel User I/O D-type connector) summed with the voltage set using the SetVoltOutput method. The values can be 'bitwise ord' to sum the software source with either or both of the other source options.	word

Example: Set the input source to software and potentiometer.

TX 52, 06, 04, 00, D0, 01, 01, 00, 02, 00,

Header: 52, 06, 04, 00, D0, 01: SetVoltsSrc, 04 byte data packet, Generic USB Device.
Chan Ident: 01, 00: Channel 1
VoltSrc: *02, 00:* selects software and potentiometer inputs

REQUEST:

Command structure (6 bytes):

0	1	2	3	4	5
header only					
53	06	Chan Ident	00	d	s

GET:

Response structure (10 bytes)
6 byte header followed by 4 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9
header						*Data*			
54	06	04	00	d|	s	Chan Ident		VoltsSrc	

For structure see SET message above.

MGMSG_PZ_SET_PICONSTS 0x0655
MGMSG_PZ_REQ_PICONSTS 0x0656
MGMSG_PZ_GET_PICONSTS 0x0657

Function: Used to set the proportional and integration feedback loop constants. These parameters determine the response characteristics when operating in closed loop mode.

The processors within the controller compare the required (demanded) position with the actual position to create an error, which is then passed through a digital PI-type filter. The filtered value is used to develop an output voltage to drive the piezo.

SET:

Command structure (12 bytes)

6 byte header followed by 6 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
55	06	06	00	d|	s	Chan Ident		PropConst		IntConst	

Data Structure:

field	description	format
Chan Ident	The channel being addressed	word
PropConst	The value of the proportional term in the range 0 to 255.	word
IntConst	The value of the Integral term.in the range 0 to 255	word

Example: Set the PI constants for a TPZ001 unit.

TX 55, 06, 06, 00, D0, 01, 01, 00, 64, 00, 0F, 00

Header: 55, 06, 05, 00, D0, 01: SetPIConsts, 06 byte data packet, Generic USB Device.

Chan Ident: 01, 00: Channel 1

PropConst: *64, 00:* sets the proportional constant to 100

IntConst: *0F, 00:* sets the integral constant to15

REQUEST:

Command structure (6 bytes):

0	1	2	3	4	5
header only					
56	06	Chan Ident	00	d	s

GET:

Response structure (12 bytes)

6 byte header followed by 6 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
57	06	06	00	d|	s	Chan Ident		PropConst		IntConst	

For structure see SET message above.

MGMSG_PZ_REQ_PZSTATUSBITS 0x065B
MGMSG_PZ_GET_PZSTATUSBITS 0x065C

Function: Returns a number of status flags pertaining to the operation of the piezo controller channel specified in the Chan Ident parameter. These flags are returned in a single 32 bit integer parameter and can provide additional useful status information for client application development. The individual bits (flags) of the 32 bit integer value are described in the following tables.

REQUEST:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
5B	06	Chan Ident	00	d	s

GET:
Response structure (12 bytes)
6 byte header followed by 6 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
5C	06	06	00	d|	s	Chan Ident		StatusBits			

Data Structure:

field	description	format
Chan Ident	The channel being addressed	word
StatusBits	The status bits for the associated controller channel. The meaning of the individual bits (flags) of the 32 bit integer value will depend on the controller and are described in the following tables.	dword

TPZ001 controller

Hex Value	Bit Number	Description
0x00000001	1	Piezo actuator connected (1 - connected, 0 - not connected).
	2 to 4	For Future Use
0x00000010	5	Piezo channel has been zero'd (1 - zero'd, 0 not zero'd).
0x00000020	6	Piezo channel is zeroing (1 - zeroing, 0 - not zeroing).
0x00000040	7 to 8	For Future Use
0x00000100	9	Strain gauge feedback connected (1 - connected, 0 - not connected).
	10	For Future Use
0x00000400	11	Position control mode (1 - closed loop, 0 - open loop).
	12 to 20	For Future Use

BPC series controllers

Hex Value	Bit Number	Description
0x00000001	1	Piezo actuator connected (1 - connected, 0 - not connected).
	2 to 4	For Future Use
0x00000010	5	Piezo channel has been zero'd (1 - zero'd, 0 not zero'd).
0x00000020	6	Piezo channel is zeroing (1 - zeroing, 0 - not zeroing).
0x00000040	7 to 8	For Future Use
0x00000100	9	Strain gauge feedback connected (1 - connected, 0 - not connected).
	10	For Future Use
0x00000400	11	Position control mode (1 - closed loop, 0 - open loop).
	12	For Future Use
Note. Bits 13, 14 and 15 are applicable only to BPC30x series controllers.		
0x00001000	13	Hardware set to 75 V max output voltage
0x00002000	14	Hardware set to 100 V max output voltage
0x00004000	15	Hardware set to 150 V max output voltage
	16 to 20	For Future Use
Note. Bits 21 to 28 (Digital Input States) are only applicable if the associated digital input is fitted to your controller – see the relevant handbook for more details		
0x00100000	21	Digital input 1 state (1 - logic high, 0 - logic low).
0x00200000	22	Digital input 2 state (1 - logic high, 0 - logic low).
0x00400000	23	Digital input 3 state (1 - logic high, 0 - logic low).
0x00800000	24	Digital input 4 state (1 - logic high, 0 - logic low).
0x01000000	25	Digital input 5 state (1 - logic high, 0 - logic low).
0x02000000	26	Digital input 6 state (1 - logic high, 0 - logic low).
0x04000000	27	Digital input 7 state (1 - logic high, 0 - logic low).
0x08000000	28	Digital input 8 state (1 - logic high, 0 - logic low).
	29	For Future Use
0x20000000	30	Active (1 – indicates unit is active, 0 – not active)
0x40000000	31	For Future Use
0x80000000	32	Channel enabled (1 – enabled, 0- disabled)

MGMSG_PZ_GET_PZSTATUSUPDATE 0x0661

Function: This function is used in applications where spontaneous status messages (i.e. messages sent using the START_STATUSUPDATES command) must be avoided.
Status update messages contain information about the position and status of the controller (for example position and O/P voltage). The messages will be sent by the controller each time the function is called.

GET:
Status update messages are received with the following format:-

Response structure (16 bytes)
6 byte header followed by 10 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
91	04	0E	00	d|	s	Chan Ident		OPVoltage		Position	

12	13	14	15
Status Bits			

Data Structure:

field	description	format
Chan Ident	The channel being addressed is always P_MOD_CHAN1 (0x01) encoded as a 16-bit word (0x01 0x00)	word
OPVoltage	The output voltage applied to the piezo. The voltage is returned in the range -32768 to 32767 (-7FFF to 7FFF) which corresponds to -100% to 100% of the maximum output voltage as set using the TPZ_IOSETTINGS command.	short
Position	The position of the piezo. The position is returned in the range 0 to 32767 (0 to 7FFF) which corresponds to 0 to 100% of the maximum position.	short
Status Bits	The meaning of the individual bits (flags) of the 32 bit integer value will depend on the controller and are described in the following tables.	dword

TPZ001 controller

Hex Value	Bit Number	Description
0x00000001	1	Piezo actuator connected (1 - connected, 0 - not connected).
	2 to 4	For Future Use
0x00000010	5	Piezo channel has been zero'd (1 - zero'd, 0 not zero'd).
0x00000020	6	Piezo channel is zeroing (1 - zeroing, 0 - not zeroing).
0x00000040	7 to 8	For Future Use
0x00000100	9	Strain gauge feedback connected (1 - connected, 0 - not connected).
	10	For Future Use
0x00000400	11	Position control mode (1 - closed loop, 0 - open loop).
	12 to 20	For Future Use

BPC series controllers

Hex Value	Bit Number	Description
0x00000001	1	Piezo actuator connected (1 - connected, 0 - not connected).
	2 to 4	For Future Use
0x00000010	5	Piezo channel has been zero'd (1 - zero'd, 0 not zero'd).
0x00000020	6	Piezo channel is zeroing (1 - zeroing, 0 - not zeroing).
0x00000040	7 to 8	For Future Use
0x00000100	9	Strain gauge feedback connected (1 - connected, 0 - not connected).
	10	For Future Use
0x00000400	11	Position control mode (1 - closed loop, 0 - open loop).
	12 to 20	For Future Use
Note. Bits 21 to 28 (Digital Input States) are only applicable if the associated digital input is fitted to your controller – see the relevant handbook for more details		
0x00100000	21	Digital input 1 state (1 - logic high, 0 - logic low).
0x00200000	22	Digital input 2 state (1 - logic high, 0 - logic low).
0x00400000	23	Digital input 3 state (1 - logic high, 0 - logic low).
0x00800000	24	Digital input 4 state (1 - logic high, 0 - logic low).
0x01000000	25	Digital input 5 state (1 - logic high, 0 - logic low).
0x02000000	26	Digital input 6 state (1 - logic high, 0 - logic low).
0x04000000	27	Digital input 7 state (1 - logic high, 0 - logic low).
0x08000000	28	Digital input 8 state (1 - logic high, 0 - logic low).
	29	For Future Use
0x20000000	30	Active (1 – indicates unit is active, 0 – not active)
0x40000000	31	For Future Use
0x80000000	32	Channel enabled (1 – enabled, 0- disabled)

MGMSG_PZ_ACK_PZSTATUSUPDATE 0x0662

Only Applicable If Using USB COMMS. Does not apply to RS-232 COMMS

Function: If using the USB port, this message called “server alive” must be sent by the server to the controller at least once a second or the controller will stop responding after ~50 commands.
The controller keeps track of the number of "status update" type of messages (e.g.move complete message) and it if has sent 50 of these without the server sending a "server alive" message, it will stop sending any more "status update" messages.
This function is used by the controller to check that the PC/Server has not crashed or switched off. There is no response.

Structure (6 bytes):

0	1	2	3	4	5
header only					
62	06	00	00	d	s

TX 62, 06, 00, 00, 50, 01

MGMSG_PZ_SET_OUTPUTLUT 0x0700
MGMSG_PZ_REQ_OUTPUTLUT 0x0701
MGMSG_PZ_GET_OUTPUTLUT 0x0702

Function: It is possible to use the controller in an arbitrary Waveform Generator Mode (WGM). Rather than the unit outputting an adjustable but static voltage or position, the WGM allows the user to define a voltage or position sequence to be output, either periodically or a fixed number of times, with a selectable interval between adjacent samples.

This waveform generation function is particularly useful for operations such as scanning over a particular area, or in any other application that requires a predefined movement sequence.

The waveform is stored as values in an array, with a maximum of 8000 samples per channel. The samples can have the meaning of voltage or position; if open loop operation is specified when the samples are output, then their meaning is voltage and vice versa, if the channel is set to closed loop operation, the samples are interpreted as position values. If the waveform to be output requires less than 8000 samples, it is sufficient to download the desired number of samples.

This function is used to load the LUT array with the required output waveform. The applicable channel is specified by the Chan Ident parameter

If only a sub set of the array is being used (as specified by the cyclelength parameter of the SetOutputLUTParams function), then only the first cyclelength values need to be set. In this manner, any arbitrary voltage waveform can be programmed into the LUT.

Note. The LUT values are output by the system at a maximum bandwidth of 7KHz, e.g.500 LUT values will take approximately 71 ms to be clocked out and the full 8000 LUT values will take approximately 1.14 secs.

SET:

Command structure (12 bytes)

6 byte header followed by 6 byte data packet as follows:

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td></tr>
<tr><td colspan="6"><i>header</i></td><td colspan="6"><i>Data</i></td></tr>
<tr><td>00</td><td>07</td><td>06</td><td>00</td><td>d|</td><td>s</td><td colspan="2">Chan Ident</td><td colspan="2">Index</td><td colspan="2">Output</td></tr>
</table>

Data Structure:

field	description	format
Chan Ident	The channel being addressed	word
Index	The position in the array of the value to be set (0 to 7999 for BPC, 0 to 512 for TPZ).	word
Output	The voltage value to be set. Values are set in the range -32768 to 32767 which corresponds to -100% to 100% of the max HV output (piezo drive voltage).	short

Example: Set output LUT value of 10V (for 150V piezo) in array position 2.

TX 00, 07, 06, 00, D0, 01, 01, 00, 02, 00, 88, 08

Header: 00, 07, 06, 00, D0, 01: SETOUTPUTLUT, 06 byte data packet, Generic USB Device.
Chan Ident: 01, 00: Channel 1
Index: *02, 00:* sets the value of array position 2
IntConst: *88, 08:* sets the value to 10V. (i.e. 150/10=15, 32767/15=2184, 2184=0888H)

REQUEST:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
01	07	Chan Ident	00	d	s

GET:
Response structure (12 bytes)
6 byte header followed by 6 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
02	07	06	00	d|	s	Chan Ident		Index		Output	

For structure see SET message above.

MGMSG_PZ_SET_OUTPUTLUTPARAMS 0x0703
MGMSG_PZ_REQ_OUTPUTLUTPARAMS 0x0704
MGMSG_PZ_GET_OUTPUTLUTPARAMS 0x0705

Function: It is possible to use the controller in an arbitrary Waveform Generator Mode (WGM). Rather than the unit outputting an adjustable but static voltage or position, the WGM allows the user to define a voltage or position sequence to be output, either periodically or a fixed number of times, with a selectable interval between adjacent samples.

This waveform generation function is particularly useful for operations such as scanning over a particular area, or in any other application that requires a predefined movement sequence.

This function is used to set parameters which control the output of the LUT array.

SET:

Command structure (36 bytes)

6 byte header followed by 30 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
03	07	1E	00	d|	s	Chan Ident		Mode		CycleLength	

12	13	14	15	16	17	18	19	20	21	22	23
Data											
NumCycles				DelayTime				PreCycleRest			

24	25	26	27	28	29	30	31	32	33	34	35
Data											
PostCycleRest				OPTrigStart		OPTrigWidth				TrigRepCycle	

Data Structure:

field	description	format
Chan Ident	The channel being addressed	word
Mode	Specifies the output mode of the LUT waveform as follows. Values can be 'bitwise or'd together as required. 0x01 - OUTPUTLUT_CONTINUOUS – The waveform is output continuously (i.e. until a StopOPLUT command is received). 0x02 - OUTPUTLUT_FIXED – A fixed number of waveform cycles are output (as specified in the NumCycles parameter). The following values are not applicable to the TPZ001 unit because it has no triggering functionality. 0x04 - OUTPUTLUT_OUTPUTTRIG – Enables Output Triggering. With OP Triggering enabled, the system can be configured to generate one or more hardware trigger pulses during a LUT (waveform) cycle output, as specified in the OPTrigStart parameter below.	word

	0x08 - OUTPUTLUT_INPUTTRIG –Enables Input Triggering. With INPUTTRIG set to ‘False’, the waveform generator will start as soon as it receives a StartOPLUT command. If however, INPUTTRIG is set to ‘True, waveform generation will only start if a software command is received AND the trigger input is in its active state. In most cases, the trigger input will be used to synchroize waveform generation to an external event. In this case, the StartOPLUT command can be viewed as a command to "arm" the waveform generator and the waveform will start as soon as the input becomes active. The trigger input can be used to trigger a single channel or multiple channels. In this latter case ensure that input triggering is enabled on all the desired channes. Using the trigger input for multiple channels is particularly useful to synchronize all channels to the same event. 0x10 - OUTPUTLUT_OUTPUTTRIG_SENSE_HI – determines the voltage sense and edge of the O/P trigger. If this bit is set, the units responds to a rising edge (OV to 5V) trigger. If not set it responds to a falling edge (5V to 0V). 0x20 - OUTPUTLUT_INPUTTRIG_SENSE_HI – determines the voltage sense and edge of the I/P trigger. If this bit is set, the units responds to a rising edge (OV to 5V) trigger. If not set it responds to a falling edge (5V to 0V). 0x40 - OUTPUTLUT_LUTGATED – If set to ‘1’ the trigger acts as a gate, if set to ‘0’ acts as trigger. 0x80 - OUTPUTLUT_OUTPUTTRIG_REPEAT – This parameter is a flag which determines if repeated O/P triggering is enabled. If set, the output trigger is repeated by the interval set in the TrigRepeatCycle parameter. This is useful for multiple triggering during a single voltage O/P sweep.	
CycleLength	Specifies how many samples will be output in each cycle of the waveform. It can be set in the range 0 to 7999 for BPC and MPZ units, and 0 to 512 for TPZ units. It must be less than or equal to the total number of samples that were loaded. (To set the LUT array values for a particular channel, see the SetOutputLUT function).	word
NumCycles	Specifies the number of cycles (1 to 2147483648) to be output when the Mode parameter is set to fixed. If Mode is set to Continuous, the NumCycles parameter is ignored. In both cases, the waveform is not output until a StartOPLUT command is received.	long
DelayTime	Specifies the delay (in sample intervals) that the system waits after setting each LUT output value. By default, the time the system takes to output LUT values (sampling interval) is set at the maximum bandwidth possible, i.e. 7KHz (0.14 ms) for MPZ models, 1kHz*1.0 ms) for BPC and 4 kHz (0.25 ms) for TPZ units. The DelayTime parameter specifies the time interval between neighbouring samples, i.e. for how long the	long

	sample will remain at its present value. To increase the time between samples, set the DelayTime parameter to the required additional delay (1 to 2147483648 sample intervals). In this way, the user can stretch or shrink the waveform without affecting its overall shape.	
PreCycleRest	In some applications, during waveform generation the first and the last samples may need to be handled differently from the rest of the waveform. For example, in a positioning system it may be necessary to start the movement by staying at a certain position for a specified length of time, then perform a movement, then remain at the last position for another specified length of time. This is the purpose of PreCycleRest and PostCycleRest parameters, i.e. they specify the length of time that the first and last samples are output for, independently of the DelayTime parameter. The PreCycleRest parameter allows a delay time to be set before the system starts to clock out the LUT values. The delay can be set between 0 and 2147483648 sample intervals. The system then outputs the first value in the LUT until the PreCycleRest time has expired.	long
PostCycleRest	In a similar way to PreCycleRest, the PostCycleRest parameter specifies the delay imposed by the system after a LUT table has been output. The delay can be set between 0 and 2147483648 sample intervals. The system then outputs the last value in the cycle until the PostCycleRest time has expired.	long
OPTrigStart	Output triggering is enabled by setting the value 0x04 in the MODE parameter. With Op Triggering enabled, the system can be configured to generate one or more hardware trigger pulses during a LUT (waveform) cycle output. The OPTrigStart parameter specifies the LUT value (position in the LUT array) at which to initiate an output trigger. In this way, it is possible to synchronize an output trigger with the output of a particular voltage value. Values are set in the range 1 to 8000 but must also be less than the CycleLength parameter.	word
OPTrigWidth	sets the width of the output trigger. Values are entered in 1ms increments for BPC20x models.	long
TrigRepeatCycle	specifies the repeat interval between O/P triggers when OUTPUTTRIG_REPEAT is set to True. This parameter is specified in the number of LUT values between triggers (0 to 7999 for MPZ and BPC units, 0 to 512 for TPZ units). If this value is greater than the lCycleLength parameter (set in the SetOPLUTParams method) then by definition, a repeated trigger will not occur during a single waveform cycle output.	word

Example: Set output LUT parameters as follows:
Channel: 1
Mode: OUTPUTLUT continuous
CycleLength: 40
NumCycles: 20
DelayTime: 10
PreCycleRest: 10
PostCycleRest: 10
OPTrigStart: 0
OPTrigWidth: 1
TrigRepeatCycle: 100
TX 03, 07, 1E, 00, D0, 01, 01, 00, 01, 00, 28, 00, 14, 00, 00, 00, 0A, 00, 00, 00, 0A, 00, 00, 00, 0A, 00, 00, 00, 00, 00, 01, 00, 00, 00, 64, 00

Header: 03, 07, 06, 00, D0, 01: SETOUTPUTLUTPARAMS, 30 byte data packet, Generic USB Device.
Channel: 1
Mode: OUTPUTLUT continuous
CycleLength: 00, 28
NumCycles: 00, 00, 00, 14
DelayTime: 00, 00, 00, 0A
PreCycleRest: 00, 00, 00, 0A
PostCycleRest: 00, 00, 00, 0A
OPTrigStart: 00, 00
OPTrigWidth: 00, 00, 00, 01
TrigRepeatCycle: 00, 64

REQUEST:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
04	07	Chan Ident	00	d	s

GET:
Response structure (36 bytes)
6 byte header followed by 30 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
03	07	1E	00	d|	s	Chan Ident		Mode		CycleLength	

12	13	14	15	16	17	18	19	20	21	22	23
Data											
NumCycles				DelayTime				PreCycleRest			

24	25	26	27	28	29	30	31	32	33	34	35
Data											
PostCycleRest				OPTrigStart		OPTrigWidth				TrigRepCycle	

For structure see SET message above.

MGMSG_PZ_START_LUTOUTPUT 0x0706

Function: This function is used to start the voltage waveform (LUT) outputs. Note. If the IPTrig flag of the SetOPLUTTrigParams function is set to false, this method initiates the waveform immediately. If the IPTrig flag is set to true, then this method 'arms' the system, in readiness for receipt of an input trigger.

TX structure (6 bytes):

0	1	2	3	4	5
header only					
06	07	Chan Ident	00	d	s

MGMSG_PZ_STOP_LUTOUTPUT 0x0707

Function: This function is used to stop the voltage waveform (LUT) outputs.

TX structure (6 bytes):

0	1	2	3	4	5
header only					
07	07	Chan Ident	00	d	s

MGMSG_PZ_SET_EEPROMPARAMS 0x07D0

Function: Used to save the parameter settings for the specified message. These settings may have been altered either through the various method calls or through user interaction with the GUI (specifically, by clicking on the 'Settings' button found in the lower right hand corner of the user interface).

SET:

Command structure (10 bytes)
6 byte header followed by 4 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9
header						*Data*			
D0	07	04	00	d|	s	Chan Ident		MsgID	

Data Structure:

field	description	format
Chan Ident	The channel being addressed	word
MsgID	The message ID of the message containing the parameters to be saved.	word

Example:

TX D0, 07, 04, 00, D0, 01, 01, 00, 03, 07,

Header: D0, 07, 04, 00, D0, 01: Set_EEPROMPARAMS, 04 byte data packet, Generic USB Device.
Chan Ident: 01, 00: Channel 1
MsgID: Save parameters specified by message 0703 (SetOutputLUTParams).

MGMSG_PZ_SET_TPZ_DISPSETTINGS 0x07D1
MGMSG_PZ_REQ_TPZ_DISPSETTINGS 0x07D2
MGMSG_PZ_GET_TPZ_DISPSETTINGS 0x07D3

Function: Used to set the intensity of the LED display on the front of the TPZ unit.

SET:

Command structure (8 bytes)

6 byte header followed by 2 byte data packet as follows:

0	1	2	3	4	5	6	7
header						*Data*	
D1	07	02	00	d|	s	DispIntensity	

Data Structure:

field	description	format
DispIntensity	The intensity is set as a value from 0 (Off) to 255 (brightest).	word

Example: Set the input source to software and potentiometer.

TX D1, 07, 02, 00, D0, 01, 64, 00,

Header: D1, 07, 02, 00, D0, 01: Set_DISPSETTINGS, 02 byte data packet, Generic USB Device.
DispIntensity: 64, 00: Sets the display brightness to 100 (40%)

REQ:

Command structure (6 bytes):

0	1	2	3	4	5
header only					
D2	07	01	00	d	s

Example: Request the display intensity

TX D2, 07, 01, 00, 50, 01

GET:

Command structure (8 bytes)

6 byte header followed by 2 byte data packet as follows:

0	1	2	3	4	5	6	7
header						*Data*	
D3	07	02	00	d|	s	DispIntensity	

See SET for data structure.

MGMSG_PZ_SET_TPZ_IOSETTINGS 0x07D4
MGMSG_PZ_REQ_TPZ_IOSETTINGS 0x07D5
MGMSG_PZ_GET_TPZ_IOSETTINGS 0x07D6

Function: This function is used to set various I/O settings as described below. The settings can be saved (persisted) to the EEPROM by calling the MGMSG_PZ_SET_EEPROMPARAMS function.

SET:

Command structure (16 bytes)

6 byte header followed by 10 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
D4	07	0E	00	d|	s	Chan Ident		VoltageLimit		HubAnalogIP	

12	13	14	15
Data			
Future Use		Future Use	

Data Structure:

field	description	format
Chan Ident	The channel being addressed is always P_MOD_CHAN1 (0x01) encoded as a 16-bit word (0x01 0x00)	word
VoltageLimit	The piezo actuator connected to the T-Cube has a specific maximum operating voltage range. This parameter sets the maximum output to the value specified as follows: 0x01 VOLTAGELIMIT_75V 75V limit 0x02 VOLTAGELIMIT_100V 100V limit 0x03 VOLTAGELIMIT_150V 150V limit	word
HubAnalogInput	When the T-Cube Piezo Driver unit is used in conjunction with the T-Cube Strain Gauge Reader (TSG001) on the T-Cube Controller Hub (TCH001), a feedback signal can be passed from the Strain Gauge Reader to the Piezo unit. High precision closed loop operation is then possible using our complete range of feedback-equipped piezo actuators. This parameter is used to select the way in which the feedback signal is routed to the Piezo unit as follows: 0x01 HUB_ANALOGUEIN_A the feedback signals run through all T-Cube bays. 0x02 HUB_ANALOGUEIN_B the feedback signals run between adjacent pairs of T-Cube bays (i.e. 1&2, 3&4, 5&6). This setting is useful when several pairs of Strain Gauge/Piezo Driver cubes are being used on the same hub. 0x03 EXTSIG_SMA the feedback signals run through the rear panel SMA connectors.	word

REQ:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
D5	07	01	00	d	s

GET:
Response structure (16 bytes)
6 byte header followed by 10 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
D4	07	0E	00	d|	s	Chan Ident		VoltageLimit		HubAnalogIP	

12	13	14	15
Data			
Future Use		Future Us	

See SET message for structure.

MGMSG_PZ_SET_ZERO 0x0658

Function: This function applies a voltage of zero volts to the actuator associated with the channel specified by the lChanID parameter, and then reads the position. This reading is then taken to be the zero reference for all subsequent position readings. This routine is typically called during the initialisation or re-initialisation of the piezo arrangement.

TX structure (6 bytes):

0	1	2	3	4	5
header only					
58	06	Chan Ident	00	d	s

MGMSG_PZ_REQ_MAXTRAVEL 0x0650
MGMSG_PZ_GET_MAXTRAVEL 0x0651

Function: In the case of actuators with built in position sensing, the Piezoelectric Control Unit can detect the range of travel of the actuator since this information is programmed in the electronic circuit inside the actuator. This function retrieves the maximum travel for the piezo actuator associated with the channel specified by the Chan Ident parameter, and returns a value (in microns) in the Travel parameter.

REQ:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
50	06	01	00	d	s

Example: Request the max travel of the actuator associated with Channel 1, bay 2 (0x22)

TX 50, 06, 01, 00, 22, 01

GET:
Response structure (10 bytes)
6 byte header followed by 4 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9
header						*Data*			
51	06	04	00	d|	s	Chan ID		Travel	

Data Structure:

field	description	format
Chan Ident	The channel being addressed.	word
Travel	The max travel of the actuator associated with the specified channel in the range 0 to 65535 (0 to FFFF). The travel is read from a calibration resistor and is returned in real world units, steps of 100nm.	

Example: Set the input source to software and potentiometer.

TX 51, 06, 04, 00, 01, A2, 01, 00, C8, 00

Header: 51, 06, 04, 00, A2, 01: Get_Max Travel, 04 byte data packet, d=A2 (i.e. 22 ORed with 80), s=01 (PC).
Channel 1*: 01, 00*:
Travel: 00C8 (200 i.e. 20 µm)

MGMSG_PZ_SET_IOSETTINGS 0x0670
MGMSG_PZ_REQ_IOSETTINGS 0x0671
MGMSG_PZ_GET_IOSETTINGS 0x0672

Function: This function is used to set various I/O settings as described below. The settings can be saved (persisted) to the EEPROM by calling the MGMSG_PZ_SET_EEPROMPARAMS function.

SET:

Command structure (16 bytes)
6 byte header followed by 10 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
70	06	0E	00	d|	s	Chan Ident		AmpCurrentLim		AmpLPFilter	

12	13	14	15
Data			
FeedbackSig		BNCTrigORLVOut	

Data Structure:

field	description	format
Chan Ident	The channel being addressed is always P_MOD_CHAN1 (0x01) encoded as a 16-bit word (0x01 0x00)	word
AmpCurrentLim	This parameter sets the maximum current output for the HV amplifier circuit as follows: CURRENTLIMIT_100MA 0x00 CURRENTLIMIT_250MA 0x01 CURRENTLIMIT_500MA 0x02	word
AmpLPFilter	This parameter sets the value of the hardware low pass filter applied to the HV amplifier output channels. It can be used to improve stability and reduce noise on the HV outputs. It is not channel specific and the Chan Ident parameter is ignored for this particular setting. Values are set as follows: OUTPUTLPFILTER_10HZ 0x00 OUTPUTLPFILTER_100HZ 0x01 OUTPUTLPFILTER_5KHZ 0x02 OUTPUTLPFILTER_NONE 0x03	word
FeedbackSig	For future use. The feedback signal type is locked at AC (strain gauge) and cannot be changed at this time.	
BNCTrigORLVOut	The Control IO BNC connectors on the rear panel are dual function. When set to Low Voltage (LV) outputs they mirror the voltage on the Piezo drive HV connectors and can be connected to an oscilloscope for monitoring purposes. When set to Trigger mode they provide the trigger input and output connections. This function is used to set the mode of the rear panel BNC connectors as follows: BNCMODE_TRIG Trigger Output 0x0000 BNCMODE_LVOUT LV Output 0xFFFF	

REQ:

Command structure (6 bytes):

0	1	2	3	4	5
header only					
71	06	01	00	d	s

GET:

Response structure (16 bytes)

6 byte header followed by 10 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
72	06	0E	00	d|	s	Chan Ident		AmpCurrentLim		AmpLPFilter	

12	13	14	15
Data			
FeedbackSig		BNCTrigORLVOut	

See SET message for structure.

MGMSG_PZ_SET_OUTPUTMAXVOLTS 0x0680
MGMSG_PZ_REQ_OUTPUTMAXVOLTS 0x0681
MGMSG_PZ_GET_OUTPUTMAXVOLTS 0x0682

Function: The piezo actuator connected to the unit has a specific maximum operating voltage range: 75, 100 or 150 V. This function sets the maximum voltage for the piezo actuator associated with the specified channel.

SET:

Command structure (12 bytes)

6 byte header followed by 6 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
80	06	0C	00	d|	s	Chan Ident		Voltage		Flags	

Data Structure:

field	description	format
Chan Ident	The channel being addressed.	word
Voltage	This parameter sets the maximum output to the value specified, in 1/10 volt steps between 0 and 1500 (i.e. 0 to 150 V).	word
Flags	These flags tell the APT server certain parameters relating to the stage and controller combination. They are not relevant to the SET command and are only used in the GET_OUTPUTMAXVOLTS message	word

Note. When the SET_OUTPUTMAXVOLTS message is sent, a GET_OUTPUTMAXVOLTS message is automatically returned. This is to inform the server that the max output voltage has changed. Similarly, a GET_MAXTRAVEL message is also returned to tell the server the new max travel value.

Example: Set the max output voltage to 100V.

TX 80, 06, 06, 00, D0, 01, 01, 00, E8, 03, 08, 00

Header: 80, 06, 06, 00, D0, 01: Set_OutputMaxVolts, 06 byte data packet, d=D0 (i.e. 50 ORed with 80 i.e. generic USB device), s=01 (PC).
Channel 1*: 01, 00*:
Voltage: 03E8 (1000 i.e. 100V)
Flags: N/A

REQ:

Command structure (6 bytes):

0	1	2	3	4	5
header only					
81	06	01	00	d	s

GET:

Response structure (12 bytes)

6 byte header followed by 6 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
82	06	06	00	d|	s	Chan Ident		Voltage		Flags	

Data Structure:

field	description	format
Chan Ident	The channel being addressed.	word
Voltage	This parameter sets the maximum output to the value specified,either 750, 1000 or 1500 (i.e. 75, 100 or 150 V).	word
Flags	These flags tell the APT server certain parameters relating to the stage and controller combination. The meaning of the individual bits (flags) of the 16 bit integer value is as follows: 0x01 For Future Use 0x02 VOLTAGELIMIT_75V 75V limit 0x04 VOLTAGELIMIT_100V 100V limit 0x05 VOLTAGELIMIT_150V 150V limit	word

Example: Set the max output voltage to 100V.

TX 82, 06, 06, 00, D0, 01, 01, 00, E8, 03, 08, 00

Header: 80, 06, 06, 00, D0, 01: Get_MaxOutputVolts, 06 byte data packet, d=D0 (i.e. 50 ORed with 80 i.e. generic USB device), s=01 (PC).

Channel 1*: 01, 00*:

Voltage: 03E8 (1000 i.e. 100V)

Flags: 08, 00: 150 V max voltage

MGMSG_PZ_SET_TPZ_SLEWRATES 0x0683
MGMSG_PZ_REQ_TPZ_SLEWRATES 0x0684
MGMSG_PZ_GET_TPZ_SLEWRATES 0x0685

Function: When stages with delicate internal mechanisms are being driven, it is possible that sudden large changes to the drive voltage could cause damage. This function is used to limit the rate of change of the drive voltage. Different limits may be set for open loop and closed loop operating modes.

Note. The controller is loaded at the factory with default values suitable for driving legacy piezo stages. For newer generation stages, the slew rate is read in automatically. Consequently, these parameters should not require adjustment under normal operating conditions.

SET:

6 byte header followed by 6 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
83	06	06	00	d|	s	Chan Ident		SlewOpen		SlewClosed	

Data Structure:

field	description	format
Chan Ident	The channel being addressed.	word
SlewOpen	This parameter sets the maximum slew rate when operating in open loop mode. Values are set in the range 0 to 32767, where 0 disables the limit, and 1 is the slowest rate. Values are calculated in V/ms as follows: $\text{Slew Rate} = \frac{\text{Value x Max Voltage (i.e. 75, 100 or 150 V)}}{19000}$	word
SlewClosed	This parameter sets the maximum slew rate when operating in closed loop mode. Values are calculated as above	word

Example: Set the open and closed max slew rates to 10V/ms for a 150V piezo.

TX 83, 06, 06, 00, D0, 01, 01, 00, F2, 04, F2, 04

Header: 80, 06, 06, 00, D0, 01: Set_SlewRates, 06 byte data packet, d=D0 (i.e. 50 ORed with 80 i.e. generic USB device), s=01 (PC).
Channel 1*: 01, 00*:
SlewOpen: F2, 04 (10V/ms i.e. 1266 x 150 / 19000)
SlewClosed: F2, 04

REQ:

Command structure (6 bytes):

0	1	2	3	4	5
header only					
84	06	01	00	d	s

GET:

Response structure (12 bytes)
6 byte header followed by 6 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
85	06	06	00	d|	s	Chan Ident		SlewOpen		SlewClosed	

See SET message for structure.

MGMSG_MOT_SET_PZSTAGEPARAMDEFAULTS 0x0686

Function: If the system has become unstable, possibly due to multiple changes to parameter values, this message can be sent to the controller in order to reset parameters to the default values stored in the EEPROM.

TX structure (6 bytes):

0	1	2	3	4	5
header only					
58	06	Chan Ident	00	d	s

MGMSG_PZ_SET_LUTVALUETYPE: 0x0708

Function: It is possible to use the controller in an arbitrary Waveform Generator Mode (WGM). Rather than the unit outputting an adjustable but static voltage or position, the WGM allows the user to define a voltage or position sequence to be output, either periodically or a fixed number of times, with a selectable interval between adjacent samples. This waveform generation function is particularly useful for operations such as scanning over a particular area, or in any other application that requires a predefined movement sequence.

The waveform is stored as values in an array, with a maximum of 8000 samples per channel. The samples can have the meaning of voltage or position; if open loop operation is specified when the samples are output, then their meaning is voltage and vice versa, if the channel is set to closed loop operation, the samples are interpreted as position values. If the waveform to be output requires less than 8000 samples, it is sufficient to download the desired number of samples.

This message specifies whether the samples output from the LUT are voltage or position values.

TX structure (6 bytes):

0	1	2	3	4	5
header only					
08	07	LUTType	00	d	s

Data Structure:

field	description	format
LUTType	The LUT value type: 0x01 LUT values are Voltage 0x02 LUT values are position	char

Example: Set the LUT value type to Volts.

TX, 08,07,01,00,50,01

Notes on using this message.

This method must be called BEFORE the LUT values are downloaded.

The LUT values are scaled to either voltage or position while the LUT is being downloaded. If the value type needs to be changed during operation (e.g. the system was in open loop with volts type selected, but now needs to change to closed loop with position type) the message must be called again, and the LUT values downloaded again.

MGMSG_PZ_SET_TSG_IOSETTINGS 0x07DA
MGMSG_PZ_REQ_TSG_IOSETTINGS 0x07DB
MGMSG_PZ_GET_TSG_IOSETTINGS 0x07DC

Function: When the T-Cube Strain Gauge Reader is used in conjunction with the T-Cube Piezo Driver unit (TPZ001) on the T-Cube Controller Hub (TCH001), a feedback signal can be passed from the Strain Gauge Reader to the Piezo unit. High precision closed loop operation is then possible using our complete range of feedback-equipped piezo actuators.

This method is used to select the way in which the feedback signal is routed back to the Piezo unit.

SET:

Command structure (20 bytes)

6 byte header followed by 14 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
DA	07	0E	00	d|	s	Chan Ident		HubAnalogOP		DisplayMode	

12	13	14	15	16	17	18	19
Data							
ForceCalib				Future Use		Future Use	

Data Structure:

field	description	format
Chan Ident	The channel being addressed is always (e.g. 0x01) encoded as a 16-bit word (0x01 0x00)	word
HubAnalogueOutput	When the T-Cube Strain Gauge Reader is used in conjunction with the T-Cube Piezo Driver unit (TPZ001) on the T-Cube Controller Hub (TCH001), a feedback signal can be passed from the Strain Gauge Reader to the Piezo unit. High precision closed loop operation is then possible using our complete range of feedback-equipped piezo actuators. This message is used to select the way in which the feedback signal is routed back to the Piezo unit If set to 0x01 HUB_ANALOGUEOUT_1, the feedback signals run through all T-Cube bays. If set to 0x02 HUB_ANALOGUEOUT_2,the feedback signals run between adjacent pairs of T-Cube bays (i.e. 1&2, 3&4, 5&6). This setting is useful when several pairs of Strain Gauge/Piezo Driver cubes are being used on the same hub.	word

Display Mode	The LED display window on the front of the unit (and the display on the GUI panel) can be set to display the strain gauge signal as a position (microns), a voltage (Volts) or as a force (Newtons). This parameter sets the display mode as follows If set to 0x01 DISPUNITS_POSITION, the display shows the strain gauge signal as a position in microns. If set to 0x02 DISPUNITS_VOLTAGE, the display shows the strain gauge signal as a voltage. If set to 0x03 DISPUNITS_FORCE, the display shows the strain gauge signal as a force	word
ForceCalib	If using a force sensor with the TSG001 unit, the Force Sensor has a specific maximum operating force. This parameter sets the force calibration factor in steps of 0.001 N between 1 and 1000. The default setting for this parameter is H7530 (30,000), to be compatible with our FSC102 force sensor, which is specified to read forces up to 30N.	word

Example: Set the IO settings as follows.

TX DA, 07, 0E, 00, D0, 01, 01, 00, 01, 00, 02, 00, 30, 75, 00, 00, 00, 00, 00, 00

Header: DA, 07, 0E, 00, D0, 01: Set_TSG_IOSettings, 14 byte data packet, d=D0 (i.e. 50 ORed with 80 i.e. generic USB device), s=01 (PC).
Channel 1*: 01, 00*:
HubAnalogueOutput: 01, 00 (Hub Analogue Output A)
Display Mode: 02, 00 (Display Voltage
Force Calibration: 30, 75 30,000 x 0.001 = 30 N

REQ:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
DB	07	01	00	d	s

GET:
Response structure (16 bytes)
6 byte header followed by 10 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
DC	07	0E	00	d|	s	Chan Ident		HubAnalogOP		DisplayMode	

12	13	14	15	16	17	18	19
Data							
ForceCalib				Future Use		Future Use	

See SET message for structure.

MGMSG_PZ_REQ_TSG_READING 0x07DD
MGMSG_PZ_GET_TSG_READING 0x07DE

Function: This message returns the current reading of the strain gauge The units applicable are dependent on the current operating mode (set using the DisplayMode parameter of the SET_TSG_IOSETTINGS message.

REQUEST:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
DD	07	Chan Ident	00	d	s

GET:
Response structure (12 bytes)
6 byte header followed by 6 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
DE	07	06	00	d|	s	Chan Ident		StatusBits			

Data Structure:

field	description	format
Chan Ident	The channel being addressed	word
Reading	The current reading of the strain gauge unit. If the unit is operating in Position mode, then the returned value is a position in microns. If the unit is in Voltage mode, then the returned reading is a Voltage. If the controller is in ‘Force Sensing Mode’ then the parameter returns a force value in Newtons. Values are returned in the range -32767 to 32768, which corresponds to -100% to 100% of the maximum output as described by the Get_PZStatusUpdate message. The returned data values are sampled at 500Hz. This is particularly useful in touch probe or force sensing applications where rapid polling of the force reading is important.	short
Smoothed		word

Example: Get the readings for channel 1.

RX DE, 07, 06, 00, D0, 00, 01, 00, 52, 00, 50, 00,

Header: DE, 07, 06, 00, D0, 00: Get_TSG_Readings, 6 byte data packet, d=D0 (i.e. 50 ORed with 80 i.e. generic USB device), s=01 (PC).
Channel 1: 01, 00
Reading: 52, 00 (i.e. 82)
Smoothed: 52, 00

Message Cross Reference by Unit Part Number

This section lists the messages applicable to each controller part number

Messages Applicable to BPC20x Series

MGMSG_MOD_IDENTIFY	0x0223	16
MGMSG_MOD_SET_CHANENABLESTATE	0x0210	17
MGMSG_MOD_REQ_CHANENABLESTATE	0x0211	17
MGMSG_MOD_GET_CHANENABLESTATE	0x0212	17
MGMSG_HW_DISCONNECT	0x002H	19
MGMSG_HW_RESPONSE	0x0080	19
MGMSG_HW_START_UPDATEMSGS	0x0011	20
MGMSG_HW_STOP_UPDATEMSGS	0x0012	20
MGMSG_HW_REQ_INFO	0x005H	21
MGMSG_HW_GET_INFO	0x006H	21
MGMSG_RACK_REQ_BAYUSED	0x0060	23
MGMSG_RACK_GET_BAYUSED	0x0061	23
MGMSG_RACK_REQ_STATUSBITS	0x0226	25
MGMSG_RACK_GET_STATUSBITS	0x0227	25
MGMSG_RACK_SET_DIGOUTPUTS	0x0228	26
MGMSG_RACK_REQ_DIGOUTPUTS	0x0229	26
MGMSG_RACK_GET_DIGOUTPUTS	0x0230	26
MGMSG_PZ_SET_POSCONTROLMODE	0x0640	94
MGMSG_PZ_REQ_POSCONTROLMODE	0x0641	94
MGMSG_PZ_GET_POSCONTROLMODE	0x0642	94
MGMSG_PZ_SET_OUTPUTVOLTS	0x0643	96
MGMSG_PZ_REQ_OUTPUTVOLTS	0x0644	96
MGMSG_PZ_GET_OUTPUTVOLTS	0x0645	96
MGMSG_PZ_SET_OUTPUTPOS	0x0646	97
MGMSG_PZ_REQ_OUTPUTPOS	0x0647	97
MGMSG_PZ_GET_OUTPUTPOS	0x0648	97
MGMSG_PZ_SET_INPUTVOLTSSRC	0x0652	98
MGMSG_PZ_REQ_INPUTVOLTSSRC	0x0653	98
MGMSG_PZ_GET_INPUTVOLTSSRC	0x0654	98
MGMSG_PZ_SET_PICONSTS	0x0655	100
MGMSG_PZ_REQ_PICONSTS	0x0656	100
MGMSG_PZ_GET_PICONSTS	0x0657	100
MGMSG_PZ_REQ_PZSTATUSBITS	0x065B	101
MGMSG_PZ_GET_PZSTATUSBITS	0x065C	101
MGMSG_PZ_GET_PZSTATUSUPDATE	0x0661	103
MGMSG_PZ_SET_OUTPUTLUT	0x0700	106
MGMSG_PZ_REQ_OUTPUTLUT	0x0701	106
MGMSG_PZ_GET_OUTPUTLUT	0x0702	106
MGMSG_PZ_SET_OUTPUTLUTPARAMS	0x0703	108
MGMSG_PZ_REQ_OUTPUTLUTPARAMS	0x0704	108
MGMSG_PZ_GET_OUTPUTLUTPARAMS	0x0705	108
MGMSG_PZ_START_LUTOUTPUT	0x0706	112
MGMSG_PZ_STOP_LUTOUTPUT	0x0707	112
MGMSG_PZ_SET_ZERO	0x0658	117
MGMSG_PZ_REQ_MAXTRAVEL	0x0650	118
MGMSG_PZ_GET_MAXTRAVEL	0x0651	118
MGMSG_PZ_SET_OUTPUTMAXVOLTS	0x0680	121
MGMSG_PZ_REQ_OUTPUTMAXVOLTS	0x0681	121
MGMSG_PZ_GET_OUTPUTMAXVOLTS	0x0682	121

Messages Applicable to BPC30x Series

MGMSG_MOD_IDENTIFY	0x0223	16
MGMSG_MOD_SET_CHANENABLESTATE	0x0210	17
MGMSG_MOD_REQ_CHANENABLESTATE	0x0211	17
MGMSG_MOD_GET_CHANENABLESTATE	0x0212	17
MGMSG_HW_DISCONNECT	0x002H	19
MGMSG_HW_RESPONSE	0x0080	19
MGMSG_HW_START_UPDATEMSGS	0x0011	20
MGMSG_HW_STOP_UPDATEMSGS	0x0012	20
MGMSG_HW_REQ_INFO	0x005H	21
MGMSG_HW_GET_INFO	0x006H	21
MGMSG_RACK_REQ_BAYUSED	0x0060	23
MGMSG_RACK_GET_BAYUSED	0x0061	23
MGMSG_RACK_REQ_STATUSBITS	0x0226	25
MGMSG_RACK_GET_STATUSBITS	0x0227	25
MGMSG_RACK_SET_DIGOUTPUTS	0x0228	26
MGMSG_RACK_REQ_DIGOUTPUTS	0x0229	26
MGMSG_RACK_GET_DIGOUTPUTS	0x0230	26
MGMSG_PZ_SET_POSCONTROLMODE	0x0640	94
MGMSG_PZ_REQ_POSCONTROLMODE	0x0641	94
MGMSG_PZ_GET_POSCONTROLMODE	0x0642	94
MGMSG_PZ_SET_OUTPUTVOLTS	0x0643	96
MGMSG_PZ_REQ_OUTPUTVOLTS	0x0644	96
MGMSG_PZ_GET_OUTPUTVOLTS	0x0645	96
MGMSG_PZ_SET_OUTPUTPOS	0x0646	97
MGMSG_PZ_REQ_OUTPUTPOS	0x0647	97
MGMSG_PZ_GET_OUTPUTPOS	0x0648	97
MGMSG_PZ_SET_INPUTVOLTSSRC	0x0652	98
MGMSG_PZ_REQ_INPUTVOLTSSRC	0x0653	98
MGMSG_PZ_GET_INPUTVOLTSSRC	0x0654	98
MGMSG_PZ_SET_PICONSTS	0x0655	100
MGMSG_PZ_REQ_PICONSTS	0x0656	100
MGMSG_PZ_GET_PICONSTS	0x0657	100
MGMSG_PZ_REQ_PZSTATUSBITS	0x065B	101
MGMSG_PZ_GET_PZSTATUSBITS	0x065C	101
MGMSG_PZ_GET_PZSTATUSUPDATE	0x0661	103
MGMSG_PZ_ACK_PZSTATUSUPDATE	0x0662	105
MGMSG_PZ_SET_OUTPUTLUT	0x0700	106
MGMSG_PZ_REQ_OUTPUTLUT	0x0701	106
MGMSG_PZ_GET_OUTPUTLUT	0x0702	106
MGMSG_PZ_SET_OUTPUTLUTPARAMS	0x0703	108
MGMSG_PZ_REQ_OUTPUTLUTPARAMS	0x0704	108
MGMSG_PZ_GET_OUTPUTLUTPARAMS	0x0705	108
MGMSG_PZ_START_LUTOUTPUT	0x0706	112
MGMSG_PZ_STOP_LUTOUTPUT	0x0707	112
MGMSG_PZ_SET_ZERO	0x0658	117
MGMSG_PZ_SET_OUTPUTMAXVOLTS	0x0680	121
MGMSG_PZ_REQ_OUTPUTMAXVOLTS	0x0681	121
MGMSG_PZ_GET_OUTPUTMAXVOLTS	0x0682	121
MGMSG_PZ_SET_SLEWRATES	0x0683	123
MGMSG_PZ_REQ_SLEWRATES	0x0684	123
MGMSG_PZ_GET_SLEWRATES	0x0685	123
MGMSG_MOT_SET_PZSTAGEPARAMDEFAULTS	0x0686	125

Messages Applicable to TPZ001

Messages Applicable to TSG001

Messages Applicable to MPZ601

Messages Applicable to TDC001

Messages Applicable to TSC001

MGMSG_MOD_IDENTIFY	0x0223	16
MGMSG_MOD_SET_CHANENABLESTATE	0x0210	17
MGMSG_MOD_REQ_CHANENABLESTATE	0x0211	17
MGMSG_MOD_GET_CHANENABLESTATE	0x0212	17
MGMSG_HW_RESPONSE	0x0080	19
MGMSG_HW_START_UPDATEMSGS	0x0011	20
MGMSG_HW_STOP_UPDATEMSGS	0x0012	20
MGMSG_HW_REQ_INFO	0x005H	21
MGMSG_HW_GET_INFO	0x006H	21
MGMSG_HUB_REQ_BAYUSED	0x0065	24
MGMSG_HUB_GET_BAYUSED	0x0066	24
MGMSG_MOT_MOVE_COMPLETED	0x0464	44
MGMSG_MOT_MOVE_ABSOLUTE	0x0453	45
MGMSG_MOT_MOVE_STOP	0x0465	49
MGMSG_MOT_SET_AVMODES	0x04B3	53
MGMSG_MOT_REQ_AVMODES	0x04B4	53
MGMSG_MOT_GET_AVMODES	0x04B5	53
MGMSG_MOT_SET_BUTTONPARAMS	0x04B6	58
MGMSG_MOT_REQ_BUTTONPARAMS	0x04B7	58
MGMSG_MOT_GET_BUTTONPARAMS	0x04B8	58
MGMSG_MOT_SET_EEPROMPARAMS:	0x04B9	60
MGMSG_MOT_GET_STATUSUPDATE	0x0481	77
MGMSG_MOT_SET_SOL_OPERATINGMODE	0x04C0	85
MGMSG_MOT_REQ_SOL_OPERATINGMODE	0x04C1	85
MGMSG_MOT_GET_SOL_OPERATINGMODE	0x04C2	85
MGMSG_MOT_SET_SOL_CYCLEPARAMS	0x04C3	87
MGMSG_MOT_REQ_SOL_CYCLEPARAMS	0x04C4	87
MGMSG_MOT_GET_SOL_CYCLEPARAMS	0x04C5	87
MGMSG_MOT_SET_SOL_INTERLOCKMODE	0x04C6	89
MGMSG_MOT_REQ_SOL_INTERLOCKMODE	0x04C7	89
MGMSG_MOT_GET_SOL_INTERLOCKMODE	0x04C8	89
MGMSG_MOT_SET_SOL_STATE	0x04CB	91
MGMSG_MOT_REQ_SOL_STATE	0x04CC	91
MGMSG_MOT_GET_SOL_STATE	0x04CD	91