Thorlabs Motion Controllers

Host-Controller Communications Protocol

Date: 30 Feb 2024

Messages Applicable to K10CR1 Only

Messages Applicable to BSC10x and BSC20x

MGMSG_MOD_IDENTIFY 0x0223

Function: Instruct hardware unit to identify itself (by flashing its front panel LEDs).
In card-slot (bay) type of systems (which are usually the multi-channel controllers such as BSC102, BSC103, BPC302, BPC303, PPC102) the front panel LED that flashes in response to this command is controlled by the motherboard, not the individual channel cards. For these controllers the destination byte of the MGMSG_MOD_IDENTIFY message must be the motherboard (0x11) and the Channel Ident byte is used to select the channel to be identified. In single-channel controllers the Channel Ident byte is ignored as the destination of the command is uniquely identified by the USB serial number of the controller.

Channel Idents
0x01 channel 1
0x02 channel 2

Command structure (6 bytes):

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

Example:
Identify controller #1 (channel 1 of the BSC103 controller) by flashing its front panel LED.

TX 23, 02, 01, 00, 11, 01

Identify the TDC001 controller (possibly within a group of various Thorlabs controllers in system):

TX 23, 02, 00, 00, 50, 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

For the TIM101 4 channel controller, the following idents are also used
0x04 channel 3
0x08 channel 4

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 0x0002

Function: Sent by the hardware unit or host 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 controllers to notify Thorlabs Server of some event that requires user intervention, usually some fault or error condition that needs to be handled before normal operation can resume. The message transmits the fault code as a numerical value – see the Return Codes listed in the Thorlabs Server helpfile for details on the specific return codes.

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_RICHRESPONSE 0x0081

Function: Similarly, to HW_RESPONSE, this message is sent by the controllers to notify Thorlabs Server of some event that requires user intervention, usually some fault or error condition that needs to be handled before normal operation can resume. However, unlike HW_RESPONSE, this message also transmits a printable text string. Upon receiving the message, Thorlabs Server displays both the numerical value and the text information, which is useful in finding the cause of the problem.

REQ:

Response structure (74 bytes):
6-byte header followed by 68-byte (0x44) data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
header						*data*									
81	00	44	00	d|	s	MsgIdent		Code		<--------Notes------->				

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
data															
<----------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
data									
<----------Notes---------->									

Data structure:

field	description	format
MsgIdent	If the message is sent in response to an Thorlabs Software message, these bytes show the message number that evoked the message. Most often though the message is transmitted due to some unexpected fault condition, in which case these bytes are 0x00, 0x00	word
Code	This is an internal Thorlabs specific code that specifies the condition that has caused the message (see Return Codes).	word]
Notes	This is a zero-terminated printable (ascii) text string that contains the textual information about the condition that has occurred. For example: “Hardware Time Out Error”.	char[64 bytes]

MGMSG_HW_START_UPDATEMSGS 0x0011

Function: Sent to start automatic 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 every 100 msec 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	Unused	Unused	d	s

REQUEST: **N/A**

MGMSG_HW_STOP_UPDATEMSGS 0x0012

Function: Sent to stop automatic 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 **0x0005**
MGMSG_HW_GET_INFO **0x0006**

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>		<Firmware> Version >				<----------For internal use only---------->							

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

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

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

80	81	82	83	84	85	86	87	88	89
data									
< For internal use only -->				HW Version		Mod State		<-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
firmware version	firmware version byte[20] = minor revision number byte[21] = interim revision number byte[22] = major revision number byte[23] = unused	byte[4]
HW Version	The hardware version number	word
Mod State	The modification state of the hardware	word
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, ……., 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
firmware Version: 02, 01, 39, 00: 3735810
HW Version: 01, 00 Hardware version 01
Mod State: 03, 00, Modification stage 03.
No Chan: 01, 00: 1 active channel

MGMSG_RACK_REQ_BAYUSED **0x0060**
MGMSG_RACK_GET_BAYUSED **0x0061**

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
0x00 Bay 1
0x01 Bay 2 to
0x09 Bay 10

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

TX 60, 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 61, 00, 00, 01, 11, 01

MGMSG_HUB_REQ_BAYUSED **0x0065**
MGMSG_HUB_GET_BAYUSED **0x0066**

Function: Sent to determine to which bay a specific unit 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

RX 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 several digital inputs. This function returns several 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					
26	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 several 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 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), 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

MGMSG_MOD_SET_DIGOUTPUTS 0x0213
MGMSG_MOD_REQ_DIGOUTPUTS 0x0214
MGMSG_MOD_GET_DIGOUTPUTS 0x0215

Function: The CONTROL IO connector on the rear panel of the unit exposes several digital outputs. The number of outputs available depends on the type of unit. This message is used to configure these digital outputs.

SET:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
13	02	Bit	00	d	s

Note. On brushless DC controllers (e.g., BBD201), the digital output and trigger output use a common pin. Before calling this message to set the digital output, the trigger functionality must be disabled by calling the Set Trigger message.

The outputs are set (and returned) in the bits of the Bits parameter, input No 1 being the least significant bit and input No 4 being the most significant. The number of bits used is dependent on the number of digital outputs present on the associated hardware unit.

For example, to turn on the digital output on a BSC201 motor controller, the least significant bit of the Bits parameter should be set to 1. Similarly, to turn on all four digital outputs on a BNT001 NanoTrak unit, the bits of the Bits parameter should be set to 1111 (15), and to turn the same outputs off, the Bits should be set to 0000.

Example: Set the digital input of the BSC201 controller on:

TX 13, 02, 01, 00, 50, 01

REQ:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
14	02	Bits	00	d	s

GET:
Response structure (6 bytes):

0	1	2	3	4	5
header only					
15	02	Bit	00	d	s

For structure see SET message above.

MGMSG_HW_SET_KCUBEMMILOCK 0x0250
MGMSG_HW_REQ_KCUBEMMILOCK 0x0251
MGMSG_HW_GET_KCUBEMMILOCK 0x0252

THIS MESSAGE IS APPLICABLE ONLY TO K-CUBE NanoTrak (KNA101-IR), K-Cube Laser Source (KLS1550 and KLS635) and K-Cube Laser Diode Driver (KLD101) UNITS

Function: This message is used to lock/unlock the controls on the top panel of the K-Cube units (wheel, joystick, buttons etc). Safety features such as the power switch and laser enable are not affected by this message. The message has global effect for all channels present on a particular unit. If the MMILock byte is set to 0x01, the controls are locked, if set to 0x02 the controls are unlocked. This message is non-volatile and will reset to unlock with each power cycle.

SET:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
50	02	00	MMILock	d	s

Example: Lock the top panel controls:

TX 50, 02, 00, 01, 50, 01

REQ:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
51	02	00	MMILock	d	s

GET:
Response structure (6 bytes):

0	1	2	3	4	5
header only					
52	02	00	MMILock	d	s

For structure see SET message above.

MGMSG_RESTOREFACTORYSETTINGS 0x0686

THIS MESSAGE IS APPLICABLE ONLY TO THE FOLLOWING CONTROLLERS:
Benchtop Piezo Controllers (BPC301 and BPC303)
K-CUBE NanoTrak (KNA101-IR)
K-Cube Laser Source (KLS1550 and KLS635)
K-Cube Laser Diode Driver (KLD101) UNITS

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

TX structure (6 bytes):

0	1	2	3	4	5
header only					
86	06	Chan Ident	00	d	s

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
MST602 – 2 Channel Modular Stepper Driver (2013 onwards)
BSC101 – 1 Channel Benchtop Stepper Driver (2006 onwards)
BSC102 – 2 Channel Benchtop Stepper Driver (2006 onwards)
BSC103 – 3 Channel Benchtop Stepper Driver (2006 onwards)
BSC201 – 1 Channel Benchtop Stepper Driver (2012 onwards)
BSC202 – 2 Channel Benchtop Stepper Driver (2012 onwards)
BSC203 – 3 Channel Benchtop Stepper Driver (2012 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
TDIxxx – 2 Channel Brushless DC Motor Driver
TBD001 – 1 Channel T-Cube Brushless DC Driver
KST101 – 1 Channel K-Cube Stepper Driver
KDC101 – 1 Channel K-Cube DC Servo Driver
KSC101 – 1 Channel K-Cube Solenoid Driver
KBD101 – 1 Channel K-Cube Brushless DC 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 (on solenoid control units). 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_HW_YES_FLASH_PROGRAMMING 0x0017

Function: This message is sent by the server on start-up; however, it is a deprecated message (i.e., has no function) and can be ignored.

Command structure (6 bytes):

0	1	2	3	4	5
header only					
17	00	Unused	Unused	d	s

REQUEST: **N/A**

MGMSG_HW_NO_FLASH_PROGRAMMING 0x0018

Function: This message is sent on start up to notify the controller of the source and destination addresses. A client application must send this message as part of its initialization process.

SET:

Command structure (6 bytes):

0	1	2	3	4	5
header only					
18	00	00	00	d	s

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

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. 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 8.	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, 40, 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: 40, 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: Similarly, to the PosCounter message described previously, this message is used to set the encoder count in the controller and is only applicable to stages and actuators fitted with an encoder. In general, this command is not normally used. 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 encoder 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 8.	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, 40, 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: 40, 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					
0A	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. For DC servo controllers, the velocity is set in encoder counts/sec and acceleration is set in encoder counts/sec/sec.
For stepper motor controllers the velocity is set in microsteps/sec and acceleration is set in microsteps/sec/sec.

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. If applicable, the scaling between real time values and this parameter is detailed in Section 8.	long
Maximum Vel	The maximum (final) velocity in encoder counts /sec. 4-byte unsigned long value. If applicable, the scaling between real time values and this parameter is detailed in Section 8.	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, For DC servo controllers, values set in encoder counts. For stepper motor controllers the values is set in microsteps.

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 8.	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 8.	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 8.	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_REQ_ADCINPUTS 0x042B
MGMSG_MOT_GET_ADCINPUTS 0x042C

Function: This message reads the voltage applied to the analog input on the rear panel CONTROL IO connector and returns a value in the ADCInput1 parameter. The returned value is in the range 0 to 32768, which corresponds to zero to 5 V.
Note. The ADCInput2 parameter is not used at this time.
In this way, a 0 to 5V signal generated by a client system could be read in by calling this method and monitored by a custom client application. When the signal reaches a specified value, the application could instigate further actions, such as a motor move.

REQUEST:
Command structure (6 bytes):

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

GET:
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*			
2C	04	04	00	d|	s	ADCInput1		ADCInput2	

Data Structure:

field	description	format
ADCInput1	The voltage state of the analog input pin, in the range 0 to 32768, which corresponds to zero to 5 V.	word
ADCInput2	Not used	word

Example: Get the ADC input state

RX 2C, 04, 04, 00, A2, 01, 01, 00, 00, 00,

Header: 2B, 04, 04, 00, A2, 01: GetADCInputs, 04 byte data packet, Channel 2.
ADCInput1: 00, 80: ADC Input 1 = 5V
ADCInput2: 00, 00: Not Used r

MGMSG_MOT_SET_POWERPARAMS 0x0426
MGMSG_MOT_REQ_POWERPARAMS 0x0427
MGMSG_MOT_GET_POWERPARAMS 0x0428

Note for BSC20x, MST602 and TST101 controller users
If the controllers listed above are used with Thorlabs SoftwareServer, the ini file will typically have values set of 5 for the rest power and 30 for the move power. Although these values are loaded when the server boots only the rest power value is used. This allows the user to set the rest current as normal. The move power however is not used. The move power is set within the controller as a function of velocity. This command can be used only to set the rest power.
The command MGMSG_MOT_REQ_POWERPARAMS will return the default values or the values that were set.

Function: The power needed to hold a motor in a fixed position is much smaller than that required for a move. It is good practice to decrease the power in a stationary motor to reduce heating, and thereby minimize thermal movements caused by expansion. This message sets a reduction factor for the rest power and the move power values as a percentage of full power. Typically, move power should be set to 100% and rest power to a value significantly less than this.

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*					
26	04	06	00	d|	s	Chan Ident		RestFactor		MoveFactor	

Data Structure:

field	description	format
Chan Ident	The channel being addressed	word
RestFactor	The phase power value when the motor is at rest, in the range 1 to 100 (i.e., 1% to 100% of full power).	word
MoveFactor	The phase power value when the motor is moving, in the range 1 to 100 (i.e., 1% to 100% of full power).	word

Example: Set the phase powers for channel 2 for TST001 unit

TX 26, 04, 06, 00, A2, 01, 01, 00, 0A, 00, 64, 00

Header: 26, 04, 06, 00, A2, 01: SetPowerParams, 06 byte data packet, Channel 2.
Chan Ident: 01, 00: Channel 1 (always set to 1 for TST001)
RestFactor: 0A, 00: Set rest power to 10% of full power
MoveFactor: 64, 00: Set move power to 100% of full power

REQUEST:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
27	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*					
28	04	06	00	d|	s	Chan Ident		RestFactor		MoveFactor	

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. Currently 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 8.	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 currently 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 8.	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 currently 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 8.	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	The direction sense for a move to Home, either 1 - forward/Positive or 2 - reverse/negative.	word
Limit Switch	The limit switch associated with the home position 1 - hardware reverse or 4 - hardware forward	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 8.	long
Offset Distance	The distance of the home position from the Home Limit Switch. This is a 4 byte signed integer that specifies the offset distance in position encoder counts. The scaling between real time values and this parameter is detailed in Section 8	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. **(Not applicable to TDC001 units)**	long
CCW Soft Limit	Counter clockwise software limit in position steps (scaling as for CW limit). **(Not applicable to TDC001 units)**	long

Software Limit Mode	Software limit switch 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) **(Not applicable to TDC001 units)**	word

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.

Note. The direction of the jog move is device dependent, i.e., on some devices jog forward may be towards the home position while on other devices it could be the opposite.

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_VELPARAMS 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

Special Note for MST602 units
The MST602 is a true 2-channel controller, rather than two single channel controllers. In this case, as well as the Chan Ident parameter, the channel being addressed is also specified in the Direction parameter (byte 3). The lower 4 bit nibble of the direction parameter is used to address channel 1 and the upper 4 bit nibble is used to address channel 2.

Examples
to move channel1 forward, TX 57, 04, 01, 01,22,01
to move channel 1 backward, TX 57, 04, 01, 02,22,01

to move channel 2 forward, TX 57, 04, 02, 10,22,01
to move channel 2 backward, TX 57, 04, 02, 20,22,01

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:

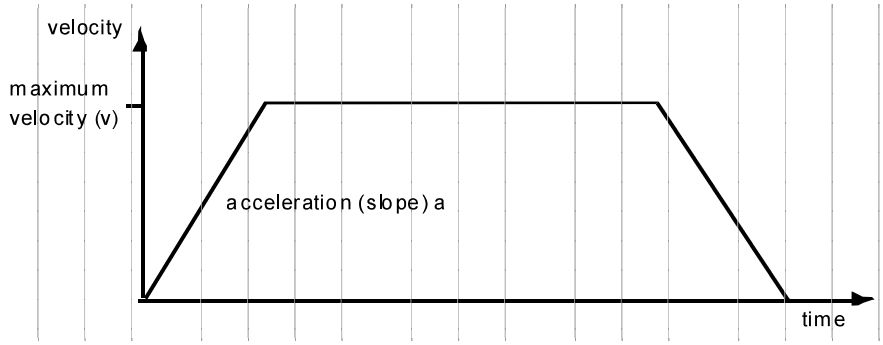
RX structure (20 bytes):

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.

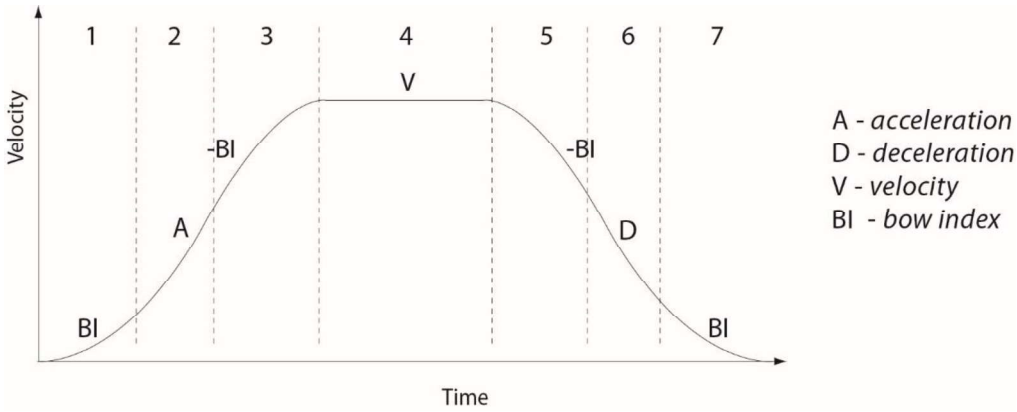
MGMSG_MOT_SET_BOWINDEX 0x04F4
MGMSG_MOT_REQ_BOWINDEX 0x04F5
MGMSG_MOT_GET_BOWINDEX 0x04F6

Function: To prevent the motor from stalling, it must be ramped up gradually to its maximum velocity. Certain limits to velocity and acceleration result from the torque and speed limits of the motor, and the inertia and friction of the parts it drives. 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.
The Bow Index parameter is used to set the profile mode to either Trapezoidal or S-curve. A Bow Index of '0' selects a trapezoidal profile. An index value of '1' to '18' selects an S-curve profile. In either case, the velocity and acceleration of the profile are specified using the Velocity Profile parameters on the Moves/Jogs tab. The Trapezoidal profile is a standard, symmetrical acceleration/deceleration motion curve, in which the start velocity is always zero. This profile is selected when the Bow Index field is set to '0'.



In a typical trapezoidal velocity profile, (see above), the stage is ramped at acceleration 'a' to a maximum velocity 'v'. As the destination is approached, the stage is decelerated at 'a' so that the final position is approached slowly in a controlled manner.

The S-curve profile is a trapezoidal curve with an additional 'Bow Value' parameter, which limits the rate of change of acceleration and smooths out the contours of the motion profile. The Bow Value is applied in mm/s^3 and is derived from the Bow Index as follows:
Bow Value = $2^{(\text{Bow Index} - 1)}$ within the range 1 to 262144 (Bow Index 1 to 18).
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.



Example
The figure above shows a typical S-curve profile. In segment (1), the S-curve profile drives the axis at the specified Bow Index (BI) until the maximum acceleration (A) is reached. The axis continues to accelerate linearly (Bow Index = 0) through segment (2). The profile then applies the negative value of Bow Index to reduce the acceleration to 0 during segment (3). The axis is now at the maximum velocity (V), at which it continues through segment (4). The profile then decelerates in a similar manner to the acceleration phase, using the Bow Index to reach the maximum deceleration (D) and then bring the axis to a stop at the destination.

Note
The higher the Bow Index, then the shorter the BI phases of the curve, and the steeper the acceleration and deceleration phases. High values of Bow Index may cause a move to overshoot.

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*			
F4	04	04	00	d|	s	Chan Ident		Bow Index	

Data Structure:

field	description	format
Chan Ident	The channel being addressed	word
BowIndex	This parameter is used to set the profile mode to either Trapezoidal or S-curve. A Bow Index of '0' selects a trapezoidal profile. An index value of '1' to '18' selects an S-curve profile.	word

Example: Set the Bow Index to 18 for Channel 1 as follows:

TX F4, 04, 04, 00, A2, 01, 01, 00, 12, 00,

Header: F4, 04, 04, 00, A2, 01: Set_BowIndex, 04 byte data packet,
Chan Ident: 01, 00: Channel 1
Bow Index: *12, 00,:* Set the Bow Index to 18

REQUEST:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
F5	04	Chan Ident	00	d	s

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

0	1	2	3	4	5	6	7	8	9
header						*Data*			
F6	04	04	00	d|	s	Chan Ident		Bow Index	

For structure see SET message above.

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					
A1	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*					
A2	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

It is not applicable for K10CR1 and K10CR2

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 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 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					
B4	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 is sprung, such that when released it returns to its central position. In this central position the motor is stationary. As the slider is moved away from the centre, 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 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 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:

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

12	13	14	15	16	17	18	19	20	21
Data									
Position1		Position2				TimeOut1		TimeOut2	

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
TimeOut1	A 'Home' move 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 button 1 must be depressed. This function is independent of the 'Mode' setting and in normal circumstances should not require adjustment. **(Not applicable to TDC001 units)**	word
TimeOut2	As TimeOut1 but for Button 2.	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					
B7	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*					
B8	04	10	00	d|	s	Chan Ident		Mode		Position1	

12	13	14	15	16	17	18	19	20	21
Data									
Position1		Position2				TimeOut1		TimeOut2	

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_POSITIONLOOPPARAMS 0x04D7
MGMSG_MOT_REQ_POSITIONLOOPPARAMS 0x04D8
MGMSG_MOT_GET_POSITIONLOOPPARAMS 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											
ILimPos				Differential		KdTimePos		KoutPos		KvffPos	

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

Data Structure:

field	description	format
Chan Ident	The channel being addressed	word
Kp Pos	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
ILimPos	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
ParamSetIx	It is possible to enter a set of PID parameters for different operating scenarios, e.g. motor is stationary, motor is accelerating, motor is at constant velocity. The specific set of PID parameters to use when the function is called is set in the ParamSetIx parameter as follows: 0 = Position PID parameters to apply when motor is stationary 1 = Position PID parameters to apply when motor is accelerating 2 = Position PID parameters to apply when motor is at constant velocity **NOTE**. This parameter is not applicable to BBD10x and BBD20x units and in this case, the units use the values from the last time the command was sent.	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
ParamSetIx: 1

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, 01, 00, 00, 00

Header: D7, 04, 1C, 00, A2, 01: Set_PositionLoopParams, 28 byte data packet, Channel 2.
Chan Ident: 01, 00: Channel 1 (always set to 1 for BBD202)
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.
ParamSetIx: 01, 00,: Use PID parameter set 1.

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_MOTOROUTPUTPARAMS 0x04DA
MGMSG_MOT_REQ_MOTOROUTPUTPARAMS 0x04DB
MGMSG_MOT_GET_MOTOROUTPUTPARAMS 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	Not implemented.	word

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 BBD202)
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_TRACKSETTLEPARAMS 0x04E0
MGMSG_MOT_REQ_TRACKSETTLEPARAMS 0x04E1
MGMSG_MOT_GET_TRACKSETTLEPARAMS 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: SetTrackSettledParams, 0CH (12) byte data packet, Channel 2.
Chan Ident: 01, 00: Channel 1 (always set to 1 for BBD202)
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_PROFILEMODEPARAMS 0x04E3
MGMSG_MOT_REQ_PROFILEMODEPARAMS 0x04E4
MGMSG_MOT_GET_PROFILEMODEPARAMS 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 BBD202)
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_JOYSTICKPARAMS 0x04E6
MGMSG_MOT_REQ_JOYSTICKPARAMS 0x04E7
MGMSG_MOT_GET_JOYSTICKPARAMS 0x04E8

Function: The MJC001 joystick console has been designed for use by microscopists to provide intuitive, tactile, manual positioning of the stage. The console consists of a two axis joystick for XY control which features both low and high gear modes. This message is used to set max velocity and acceleration values for these modes.

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*					
E6	04	14	00	d|	s	Chan Ident		JSGearLowMaxVel		

12	13	14	15	16	17	18	19	20	21	22	23
Data											
JSGearHighMaxVel				JSGearHighLowAccn				JSGearHighHighAccn			

24	25
Data	
DirSense	

Data Structure:

field	description	format
Chan Ident	The channel being addressed	word
JSGearLowMaxVel	Specifies the max velocity (in encoder counts/cycle) of a joystick move when low gear mode is selected. It accepts values in the range 0 to 4294967295. 1 mm / sec equals 134218 PMD units	long
JSGearHighMaxVel	Specifies the max velocity (in encoder counts/cycle) of a joystick move when high gear mode is selected. It accepts values in the range 0 to 4294967295. 1 mm / sec equals 134218 PMD units	long
JSGearLowAccn	Specifies the acceleration (in encoder counts/cycle) of a joystick move when low gear mode is selected. It accepts values in the range 0 to 4294967295. 1 mm /sec^2 equals 13.7439 PMD units.	long
JSGearHighAccn	Specifies the acceleration (in encoder counts/cycle) of a joystick move when high gear mode is selected. It accepts values in the range 0 to 4294967295. 1 mm /sec^2 equals 13.7439 PMD units.	long
DirSense	The actual direction sense of any joystick initiated move is dependent upon the application. This parameter can be used to reverse the sense of direction for a particular application and is useful when matching joystick direction sense to actual stage direction sense. DIRSENSE_POS 0X0001 Direction Positive DIRSENSE_NEG 0X0002 Direction Negative	word

Example: Set the joystick parameters for bay 2 as follows:
JSGearLowMaxVel: 1 mm/sec
JSGearHighMaxVel: 10 mm/sec
JSGearLowAccn: 0.5 mm /sec^2
JSGearHighAccn: 5.0 mm /sec^2
DirSens: Positive

TX E6, 04, 14, 00, A2, 01, 01, 00, 4A, 0C, 02, 00, E4, 7A, 14, 00, 07, 00, 00, 00, 46, 00, 00, 00, 01, 00

Header: E6, 04, 14, 00, A2, 01: SetJoystickParams, 14H (20) byte data packet, bay 2.
Chan Ident: 01, 00: Channel 1 (always set to 1 for BBD202)
*JSGearLowMaxVel: 4A, 0C, 02, 00 (*134218)
JSGearHighMaxVel: E4, 7A, 14, 00 (1342180)
JSGearLowAccn: 07, 00, 00, 00 (7.0)
JSGearHighAccn: 46, 00, 00, 00 (70.0)
DirSens: *01, 00*

REQUEST:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
E7	04	Chan Ident	00	d	s

GET:
Response 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*					
E8	04	14	00	d|	s	Chan Ident		JSGearLowMaxVel		

12	13	14	15	16	17	18	19	20	21	22	23
Data											
JSGearHighMaxVel				JSGearHighLowAccn				JSGearHighHighAccn			

24	25
Data	
DirSense	

For structure see SET message above.

MGMSG_MOT_SET_CURRENTLOOPPARAMS 0x04D4
MGMSG_MOT_REQ_CURRENTLOOPPARAMS 0x04D5
MGMSG_MOT_GET_CURRENTLOOPPARAMS 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		ParamSetIx		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
ParamSetIx	It is possible to enter a set of PID parameters for different operating scenarios, e.g. motor is stationary, motor is in motion or not yet settled at target position. The specific set of PID parameters to use when the function is called is set in the ParamSetIx parameter as follows: 0 = Normal current loop parameter set (motor in motion, or not yet settled at target position) 1 = Settled current loop parameter set (motor stationary, settled at target position) **NOTE**. This parameter is not applicable to BBD10x and BBD20x units and in this case, the units use the values from the last time the command was sent.	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
ParamSetIx: 1

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

Header: D4, 04, 12, 00, A2, 01: Set_CurrentLoopParams, 18 byte data packet, Channel 2.
Chan Ident: 01, 00: Channel 1 (always set to 1 for BBD202)
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
ParamSetIx: 01, 00 Use parameter set 1.

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_SETTLEDCURRENTLOOPPARAMS 0x04E9
MGMSG_MOT_REQ_SETTLEDCURRENTLOOPPARAMS 0x04EA
MGMSG_MOT_GET_SETTLEDCURRENTLOOPPARAMS 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:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
E9	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	

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_SettledCurrentLoopParams, 18 byte data packet, Channel 2.
Chan Ident: 01, 00: Channel 1 (always set to 1 for BBD202)
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_STAGEAXISPARAMS 0x04F0
MGMSG_MOT_REQ_STAGEAXISPARAMS 0x04F1
MGMSG_MOT_GET_STAGEAXISPARAMS 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	4A	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:

RX F2, 04, 4A, 00, 81, 22, 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, 00, 00

Header: F2, 04, 4A, 00, 81, 22: Get_StageAxisParams, 74 byte data packet,Bay 1.
Chan Ident: 01, 00: Channel 1 (always set to 1 for BBD202)
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_SET_TSTACTUATORTYPE 0x04FE

Function: This command is for use only with the TST101 driver, and is used to define an actuator type so that the TST driver knows the effective length of the stage. This information is used if a user wishes to home the stage to the far travel end. In this case, once the stage is homed the Thorlabs Software GUI count will be set to the far travel value. For example, in the case of a ZFS25 the user will see 25mm once homed. The TST holds this value as a number of Trinamic microsteps, which will be a function of the gearbox ratio, the lead screw pitch, and the motor type. So for example the number stored in the TST for the ZFS25 is 54613333.

SET:

Command structure (6 bytes):

0	1	2	3	4	5
header only					
FE	04	Actuator Ident	00	d	s

Actuator Idents:

ZST_LEGACY_6MM	0x20
ZST_LEGACY_13MM	0x21
ZST_LEGACY_25MM	0x22
ZST_NEW_6MM	0x30
ZST_NEW_13MM	0x31
ZST_NEW_25MM	0x32
ZFS_NEW_6MM	0x40
ZFS_NEW_13MM	0x41
ZFS_NEW_25MM	0x42
DRV013_25MM	0x50
DRV014_50MM	0x51

Example: Set the actuator type to New ZFS 13 mm Travel:

Header: FE, 04, 31, 00, 50, 01:

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 previously. In the BSC series controllers, each channel is seen as a separate controller with its own serial number and each card must be addressed separately.

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

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*					
81	04	1C	00	d|	s	Chan Ident 1		Position			

12	13	14	15	16	17	18	19	20	21
Data									
EncCount				Status Bits				Chan Ident 2	

22	23	24	25	26	27	28	29	30	31	32	33
Data											
For Future Use				For Future Use				For 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
Position	The position encoder count. In the Thorlabs Stepper Motor controllers the encoder resolution is 25,600 or 409600 counts per mm depending on the controller. Therefore a position change of 1 mm would be seen as this parameter changing by 25,600 or 409600. 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
All remaining bytes are for future use and should be ignored		

Example: Get the status update:

81, 04, 1C, 00, 81, 50, 01, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00,

Header: 81, 04, 1C, 00, 81, 50: Get_StatusUpdate, 28 byte data packet,
Chan Ident: 01, 00: Channel 1 (always set to 1 for BSC20X)
Position: 00, 00, 00, 00:
Enc Counts: *00, 00, 00, 00:* Only used with encoded stages
Status Bits: *00, 00, 00, 00,* See below for details,:
All remaining bytes are ignored

Status Bits

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					
80	04	Chan Ident	00	d	s

GET:

See previous details on MGMSG_MOT_GET_STATUSUPDATE 0x0481.

MGMSG_MOT_GET_USTATUSUPDATE 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		MotorCurrent		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 in encoder counts (controller units). The relationship between the encoder count and physical units such as millimetres or degrees depends on both the controller and the stage. The conversion factors are listed in Section 8: “Conversion between position, velocity and acceleration values in standard physical units and their equivalent Thorlabs parameters”. For example in the BBD20X series controllers used with the MLS203 stage, 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	Actual velocity in controller units. As with position, relationship between this value and physical velocity depends on the motor and controller - see section 8 for the conversion factors. For example, this conversion factor is 204.8 for the BBD20X series controllers used with the MLS203 stage, 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
Motor Current	Motor Current in mA (range -32768 to +32767). **Note**. Legacy controllers (i.e. those designed before-2020)	word

	do not return the motor current. In this case, this value is not used.	
Status Bits	The meaning of individual bits in this 32-bit variable is described below	dword

Status Bits Description

0x00000001 - P_MOT_SB_CWHARDLIMIT
0x00000002 - P_MOT_SB_CCWHARDLIMIT
Clockwise and counter-clockwise hardware limit switches. On linear stages these also correspond to the forward and reverse limit switches. (Due to the gearbox fitted in some linear stages, the clockwise and counter-clockwise directions may not match forward and reverse.)

0x00000004 - P_MOT_SB_CWSOFTLIMIT
0x00000008 - P_MOT_SB_CCWSOFTLIMIT
Clockwise and counter-clockwise software limit switches. On some controllers a software limits can be imposed on the motion, restricting it to a narrower range than the hardware limit switches.

0x00000010 - P_MOT_SB_INMOTIONCW
0x00000020 - P_MOT_SB_INMOTIONCCW
In motion, moving clockwise or counter-clockwise.

0x00000040 - P_MOT_SB_JOGGINGCW
0x00000080 - P_MOT_SB_JOGGINGCCW
Jogging, clockwise or counter-clockwise.

0x00000100 - P_MOT_SB_CONNECTED
Indicates that the motor has been recognized by the controller.

0x00000200 - P_MOT_SB_HOMING
Indicates that the motor is performing a homing move.

0x00000400 - P_MOT_SB_HOMED
Indicates that the motor has completed the homing move, the absolute position is known and therefore the position count is now valid.

0x00000800 - P_MOT_SB_INITILIZING
For 3-phase brushless motors only: the motor is performing a phase initialization procedure, attempting to establish the correct commutation phase angle. This is an essential process for brushless motors and during this process no motion related command can be responded to.

0x00001000 - P_MOT_SB_TRACKING
Actual position is within the trajectory tracking window.

0x00002000 - P_MOT_SB_SETTLED

Indicates that the motor is not moving and it is settled at the target position. The actual position has been within the target position for a specified length of time.

0x00004000 - P_MOT_SB_POSITIONERROR
Indicates that the actual position is outside the margin specified around the trajectory position. (In simple terms the motor is not where it should be.) This can occur momentarily during fast acceleration (the motor lags behind the trajectory) or when the motor is jammed, or the move is obstructed. Typically the condition can trigger the controller to disable the motor in order to prevent damage, which in turn will clear the error.
0x00008000 - P_MOT_SB_INSTRERROR
Only used on legacy controllers. Indicates that the motion controller unable to execute command received (for example, incompatible operating mode)

0x00010000 - P_MOT_SB_INTERLOCK
Used on controllers where there is a separate signal required to enable the motor.

0x00020000 - P_MOT_SB_OVERTEMP
Indicates that either the motor power driver electronics or the motor itself has reached its maximum operating temperature. Normally results in the motor drive getting disabled.

0x00040000 - P_MOT_SB_BUSVOLTFAULT
Indicates that the supply voltage to the motor is too low. Potential reasons include a power supply fault or wiring problem.

0x00080000 - P_MOT_SB_COMMUTATIONERROR
Only used for 3-phase brushless motors. Indicates a problem with the motor commutation and normally occurs if the phase initialization process has failed (see P_MOT_SB_INITILIZING). This is an unrecoverable fault that makes motion control impossible and can only be cleared by a power cycle.

0x00100000 - P_MOT_SB_DIGIP1
0x00200000 - P_MOT_SB_DIGIP2
0x00400000 - P_MOT_SB_DIGIP3
0x00800000 - P_MOT_SB_DIGIP4
Indicates the state of the digital inputs on those controllers with a limited small number of digital I/O lines. (If a controller has more than 4 digital inputs or if there are different configuration options, a separate command is used for reading the state of the input signals.)

0x01000000 - P_MOT_SB_OVERLOAD
Indicates a motor overload condition: can overcurrent condition (see P_MOT_SB_OVERCURRENT) has occurred for a long period of time and the motor has been used beyond its power handling capabilities. Normally results in the maximum output current being reduced or the motor being disabled.

0x02000000 - P_MOT_SB_ENCODERFAULT
Indicates an encoder fault in controllers that have encoder diagnostic capabilities (e.g. M30X, M30XY).

0x04000000 - P_MOT_SB_OVERCURRENT
Indicates that the motor current has exceeded the continuous current limit specified for the motor. This can occur temporarily during heavy load or fast acceleration conditions and under these circumstances it is normal (motors are normally tolerant of brief current spikes beyond their continuous rating). However, when it occurs over a sustained length of time, it can trigger a P_MOT_SB_OVERLOAD condition.
0x08000000 - P_MOT_SB_BUSCURRENTFAULT
Indicates that excessive current is being drawn from the motor power supply. This condition typically indicates a hard wiring fault that needs to be rectified, for example a phase-to-phase short circuit in a brushless motor.

0x10000000 - P_MOT_SB_POWEROK
Indicates that all the controller power supplies are operating normally.

0x20000000 - P_MOT_SB_ACTIVE
Normally indicates that the controller is executing a motion command.

0x40000000 - P_MOT_SB_ERROR
Indicates an error condition, either listed above or arising as a result of another abnormal condition.

0x80000000 - P_MOT_SB_ENABLED
Indicates that the motor output is enabled and the controller is in charge of maintaining the required position. When the output is disabled, the motor is not controlled by the electronics and can be moved manually, as much as the mechanical construction (such as any leadscrew and gearbox fitted) allows.
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.

See the following table for a list of status bits and applicable controllers.

Motor Controller Status Bits Applicable

Bit	Definition	TDC001	TBD001	KDC101	KBD101	M30X	M30XY	BBD20X	BBD30X
0x0000.0001	P_MOT_SB_CWHARDLIMIT	✓	✓	✓	✓	✓	✓	✓	✓
0x0000.0002	P_MOT_SB_CCWHARDLIMIT	✓	✓	✓	✓	✓	✓	✓	✓
0x0000.0004	P_MOT_SB_CWSOFTLIMIT	✓	▪	✓	▪	✓	✓	▪	▪
0x0000.0008	P_MOT_SB_CCWSOFTLIMIT	✓	▪	✓	▪	✓	✓	▪	▪
0x0000.0010	P_MOT_SB_INMOTIONCW	✓	✓	✓	✓	✓	✓	✓	✓
0x0000.0020	P_MOT_SB_INMOTIONCCW	✓	✓	✓	✓	✓	✓	✓	✓
0x0000.0040	P_MOT_SB_JOGGINGCW	✓	✓	✓	✓	✓	✓	✓	✓
0x0000.0080	P_MOT_SB_JOGGINGCCW	✓	✓	✓	✓	✓	✓	✓	✓
0x0000.0100	P_MOT_SB_CONNECTED	✓	▪	✓	✓	✓	✓	▪	✓
0x0000.0200	P_MOT_SB_HOMING	✓	✓	✓	✓	✓	✓	✓	✓
0x0000.0400	P_MOT_SB_HOMED	✓	✓	✓	✓	✓	✓	✓	✓
0x0000.0800	P_MOT_SB_INITILIZING	▪	▪	▪	▪	▪	▪	▪	✓
0x0000.1000	P_MOT_SB_TRACKING	▪	✓	▪	▪	▪	▪	✓	✓
0x0000.2000	P_MOT_SB_SETTLED	▪	✓	▪	▪	▪	▪	✓	✓
0x0000.4000	P_MOT_SB_POSITIONERROR	▪	✓	✓	✓	✓	✓	✓	✓
0x0000.8000	P_MOT_SB_INSTRERROR	▪	✓	▪	▪	▪	▪	✓	▪
0x0001.0000	P_MOT_SB_INTERLOCK	▪	✓	▪	▪	▪	▪	✓	✓
0x0002.0000	P_MOT_SB_OVERTEMP	▪	✓	▪	▪	✓	✓	✓	✓
0x0004.0000	P_MOT_SB_BUSVOLTFAULT	▪	✓	▪	▪	✓	✓	✓	✓
0x0008.0000	P_MOT_SB_COMMUTATIONERROR	▪	✓	▪	▪	▪	▪	✓	✓
0x0010.0000	P_MOT_SB_DIGIP1	✓	✓	✓	✓	✓	✓	✓	▪
0x0020.0000	P_MOT_SB_DIGIP2	▪	▪	✓	✓	✓	✓	▪	▪
0x0040.0000	P_MOT_SB_DIGIP3	▪	▪	▪	▪	▪	▪	▪	▪
0x0080.0000	P_MOT_SB_DIGIP4	▪	▪	▪	▪	▪	▪	▪	▪
0x0100.0000	P_MOT_SB_OVERLOAD	▪	✓	▪	✓	✓	✓	✓	✓
0x0200.0000	P_MOT_SB_ENCODERFAULT	▪	▪	▪	▪	✓	✓	▪	▪
0x0400.0000	P_MOT_SB_OVERCURRENT	▪	▪	▪	▪	✓	✓	▪	✓
0x0800.0000	P_MOT_SB_BUSCURRENTFAULT	▪	▪	▪	▪	✓	✓	▪	✓
0x1000.0000	P_MOT_SB_POWEROK	✓	▪	▪	✓	✓	✓	✓	✓
0x2000.0000	P_MOT_SB_ACTIVE	▪	▪	▪	✓	✓	✓	▪	✓
0x4000.0000	P_MOT_SB_ERROR	✓	✓	✓	▪	✓	✓	✓	✓
0x8000.0000	P_MOT_SB_ENABLED	✓	✓	✓	✓	✓	✓	✓	✓
wMotorCurrent parameter is used and the data is valid		▪	▪	▪	▪	✓	✓	▪	✓

MGMSG_MOT_REQ_USTATUSUPDATE 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_USTATUSUPDATE 0x0491.

MGMSG_MOT_ACK_USTATUSUPDATE 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					
92	04	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

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

MGMSG_MOT_SET_TRIGGER **0x0500**
MGMSG_MOT_REQ_TRIGGER **0x0501**
MGMSG_MOT_GET_TRIGGER **0x0502**

Function: This message is used to configure the Motor controller for triggered move operation. It is possible to configure a particular controller to respond to trigger inputs, generate trigger outputs or both respond to and generate a trigger output. When a trigger input is received, the unit can be set to initiate a move (relative, absolute or home). Similarly the unit can be set to generate a trigger output signal when a specified event (e.g move initiated) occurs. For those units configured for both input and output triggering, a move can be initiated via a trigger input while at the same time, a trigger output can be generated to initiate a move on another unit.
The trigger settings can be used to configure multiple units in a master – slave set up, thereby allowing multiple channels of motion to be synchronized. Multiple moves can then be initiated via a single software or hardware trigger command.

SET:
Command structure (6 bytes):

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

Note. This message operates differently when used with brushless DC controllers (e.g. BBD20x and TBD001) as opposed to other motor controllers as described in the following paragraphs.

All benchtop stepper controllers (BSC20x,)

field	description	format
Chan Ident	The channel being addressed	char
Mode	This parameter sets the trigger mode and move type to be initiated according to the numerical value entered in bits 0 to 7 as follows Bit 0 (0x01): TRIGIN_ENABLE set to enable physical trigger input Bit 1 (0x02): TRIGOUT_ENABLE set to enable trigger output function (mode set by BIT2 or BIT3 below) Bit 2 (0x04): TRIGOUT_MODEFOLLOW set to enable physical trigger output to mirror trig in Bit 3 (0x08): TRIGOUT_MODEMOVEEND set to enable physical trigger output, remains active (high) until move end Bit 4 (0x10): TRIG_RELMOVE set for relative move on trigger Bit 5 (0x20): TRIG_ABSMOVE set for absolute move on trigger Bit 6 (0x40): TRIG_HOMEMOVE set for home sequence on trigger	char

	Bit 7 (0x80): TRIGOUT_NOTRIGIN set to enable physical trigger output with no physical trigger in (i.e. sw initiated trigger)	

Brushless DC controllers only (BBD20x, BBD30x and TBD001)

field	description	format
Chan Ident	The channel being addressed	char
Mode	This parameter sets the trigger mode and move type according to the numerical value entered in bits 0 to 7 as follows Bit 0 (0x01): TRIGIN_HIGH The Trigger input can be configured to initiate a relative, absolute or homing home, either on the rising or falling edge of the signal driving it. As the trigger input is edge sensitive, it needs to see a logic LOW to HIGH transition ("rising edge") or a logic HIGH to LOW transition ("falling edge") for the move to be started. Additionally, the move parameters must be downloaded to the unit prior to the move using the relevant relative move or absolute move methods as described below. A move already in progress will not be interrupted; therefore external triggering will not work until the previous move has been completed. If this bit is set, the logic state is set HIGH. Bit 1 (0x02): TRIGIN_RELMOVE set to enable trigger in and initiate a relative move (specified using the latest MoveRelative or MoveRelativeEx settings) when a trigger input signal is received. Bit 2 (0x04): TRIGIN_ABSMOVE set to enable trigger in and initiate an absolute move (specified using the latest MoveAbsolute or MoveAbsoluteEx settings) when a trigger input signal is received. Bit 3 (0x08): TRIGIN_HOMEMOVE set to enable trigger in and initiate a home move (specified using the latest MoveHome settings) whan atrigger input signal is received. Bit 4 (0x10): TRIGOUT_HIGH The Trigger output can be configured to be asserted to either logic HIGH or LOW as a function of certain motion-related conditions, such as when a move is in progress (In Motion), complete (Move Complete) or reaches the constant velocity phase on its trajectory (Max Vel). The logic state of the output will remain the same for as long as the chosen condition is true. If this bit is set, the logic state is set HIGH when the following conditions are true. Bit 5 (0x20): TRIGOUT_INMOTION set to enable trigger out (triggered when in motion) Bit 6 (0x40): TRIGOUT_MOTIONCOMPLETE set to enable trigger out (triggered when motion complete) Bit 7 (0x80): TRIGOUT_MAXVELOCITY set to enable trigger out (triggered when axis at maximum velocity)	char

Example: Set the trigger mode for channel 1 of the BBD201 controller as follows:
Trigger Input Rising Edge (High)
Enable trigger input and initiate a Relative Move
Trigger Output Rising Edge (High)
Enable trigger output when move complete.

TX 00, 05, 01, 53, 50, 01

00,05 SET_TRIGGER
01, Channel 1
53, i.e. 01010011
50, destination Generic USB device
01, Source PC

REQ:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
01	05	Chan Ident	00	d	s

Example: Request the trigger mode

TX 01, 05, 01, 00, 50, 01

GET:
Response structure (6 bytes):

0	1	2	3	4	5
header only					
02	05	Chan Ident	Mode	d	s

For structure see SET message above.

MGMSG_MOT_SET_KCUBEMMIPARAMS 0x0520
MGMSG_MOT_REQ_KCUBEMMIPARAMS 0x0521
MGMSG_MOT_GET_KCUBEMMIPARAMS 0x0522

This message is applicable only to KST101, KDC101, KBD101 and BBD30x units

Function: This message is used to configure the operating parameters of the top panel wheel (Joystick).

SET
Command structure (42 bytes)
6 byte header followed by 36 byte data packet.

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
20	05	1C	00	d	s	Chan Ident		JSMode		JSMaxVel	

12	13	14	15	16	17	18	19	20	21	22	23
Data											
JSMaxVel		JSAccn				DirSense		PreSetPos1			

24	25	26	27	28	29	30	31	32	33
Data									
PreSetPos2				DispBrightness		DispTimeout		DispDimLevel	

34	35	36	37	38	39	40	41
Data							
PreSetPos3				JSSensitivity		Reserved	

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
JSMode	This parameter specifies the operating mode of the wheel/joy stick as follows: 1 Velocity Control Mode - Deflecting the wheel starts a move with the velocity proportional to the deflection. The maximum velocity (i.e. velocity corresponding to the full deflection of the joystick wheel) and acceleration are specified in the MaxVel and MaxAccn parameters. 2 Jog Mode - Deflecting the wheel initiates a jog move, using the parameters specified by the SetJogStepSize and SetJogVelParams methods. Keeping the wheel deflected repeats the move automatically after the current move has completed. 3 Go To Position Mode - Deflecting the wheel starts a move from the current position to one of the two predefined “teach” positions. The teach positions are specified in number of steps from the home position in the PresetPos1 and PresetPos2 parameters.	word
JSMaxVel	The max velocity of a move initiated by the top panel	long

	velocity wheel.	
JSAccn	The max acceleration of a move initiated by the top panel velocity wheel	long
DirSense	This parameter specifies the direction of a move initiated by the velocity wheel as follows: 0 Wheel initiated moves are disabled. Wheel used for menuing only. 1 Upwards rotation of the wheel results in a positive motion (i.e. increased position count). The following option applies only when the JSMode is set to Velocity Control Mode (1). If set to Jog Mode (2) or Go to Position Mode (3), the following option is ignored. 2 Upwards rotation of the wheel results in a negative motion (i.e. decreased position count).	word
PresetPos1	The preset position 1 when operating in go to position mode, measured in position steps from the home position.	long
PresetPos2	The preset position 2 when operating in go to position mode, measured in position steps from the home position.	long
DispBrightness	In certain applications, it may be necessary to adjust the brightness of the LED display on the top of the unit. The brightness is set as a value from 0 (Off) to 100 (brightest). The display can be turned off completely by entering a setting of zero, however, pressing the MENU button on the top panel will temporarily illuminate the display at its lowest brightness setting to allow adjustments. When the display returns to its default position display mode, it will turn off again.	word
DispTimeout	'Burn In' of the display can occur if it remains static for a long time. To prevent this, the display is automatically dimmed after the time interval specified in the DispTimeout parameter has elapsed. Set in minutes in the range 0 (never dimmed) to 480. The dim level is set in the DispDimLevel parameter below.	word
DispDimLevel	The dim level, as a value from 0 (Off) to 10 (brightest) but is also limited by the DispBrightness parameter.	word
PresetPos3	**Applicable to BBD30x Only.** The preset position 3 when operating in go to position mode, measured in position steps from the home position.	long
wJSSensitivity	**Applicable to BBD30x Only.** Joystick sensitivity 0 to 65535 representing zero to maximum sensitivity	word
wReserved		word

REQ:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
21	05	Chan Ident	00	d	s

Example: Request the settings for the top panel wheel

TX 21, 05, 01, 00, 50, 01

GET:
Response structure (6 bytes):

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
22	05	1C	00	d	s	Chan Ident		JSMode		JSMaxVel	

12	13	14	15	16	17	18	19	20	21	22	23
Data											
JSMaxVel		JSAccn				DirSense		PreSetPos1			

24	25	26	27	28	29	30	31	32	33
Data									
PreSetPos2				DispBrightness		DispTimeout		DispDimLevel	

34	35	36	37	38	39	40	41
Data							
PreSetPos3				JSSensitivity		Reserved	

For structure see SET message above.

MGMSG_MOT_SET_KCUBETRIGIOCONFIG 0x0523
MGMSG_MOT_REQ_KCUBETRIGIOCONFIG 0x0524
MGMSG_MOT_GET_KCUBETRIGIOCONFIG 0x0525

This message is applicable only to KST101, KDC101 and KBD101 units

Function: The K-Cube motor controllers have two bidirectional trigger ports (TRIG1 and TRIG2) that can be used to read an external logic signal or output a logic level to control external equipment. Either of them can be independently configured as an input or an output and the active logic state can be selected High or Low to suit the requirements of the application. Electrically the ports output 5 Volt logic signals and are designed to be driven from a 5 Volt logic.
When the port is used in the input mode, the logic levels are TTL compatible, i.e. a voltage level less than 0.8 Volt will be recognised as a logic LOW and a level greater than 2.4 Volt as a logic HIGH. The input contains a weak pull-up, so the state of the input with nothing connected will default to a logic HIGH. The weak pull-up feature allows a passive device, such as a mechanical switch to be connected directly to the input.
When the port is used as an output it provides a push-pull drive of 5 Volts, with the maximum current limited to approximately 8 mA. The current limit prevents damage when the output is accidentally shorted to ground or driven to the opposite logic state by external circuity.

Warning: do not drive the TRIG ports from any voltage source that can produce an output in excess of the normal 0 to 5 Volt logic level range. In any case the voltage at the TRIG ports must be limited to -0.25 to +5.25 Volts.

SET
Command structure (28 bytes)
6 byte header followed by 22 byte data packet.

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
23	05	0C	00	d	s	Chan Ident		Trig1Mode		Trig1Polarity	

12	13	14	15	16	17	18 to 28
Data						
Trig2Mode		Trig2Polarity		Reserved		Reserved

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
Trig1Mode	TRIG1 operating mode	word
Trig1Polarity	The active state of TRIG1 (i.e. logic high or logic low) l.	word
Trig2Mode	TRIG2 operating mode	word
Trig2Polarity	The active state of TRIG2 (i.e. logic high or logic low)	word
Reserved	Bytes 16 to 28	word

Input Trigger Modes

When configured as an input, the TRIG ports can be used as a general purpose digital input, or for triggering a relative, absolute or home move as follows:

0x00 The trigger IO is disabled
0x01 General purpose logic input (read through status bits using the MOT_GET_STATUSBITS message).
0x02 Input trigger for relative move.
0x03 Input trigger for absolute move.
0x04 Input trigger for home move.

When used for triggering a move, the port is edge sensitive. In other words, it has to see a transition from the inactive to the active logic state (Low->High or High->Low) for the trigger input to be recognized. For the same reason a sustained logic level will not trigger repeated moves. The trigger input has to return to its inactive state first in order to start the next trigger.

Output Trigger Modes

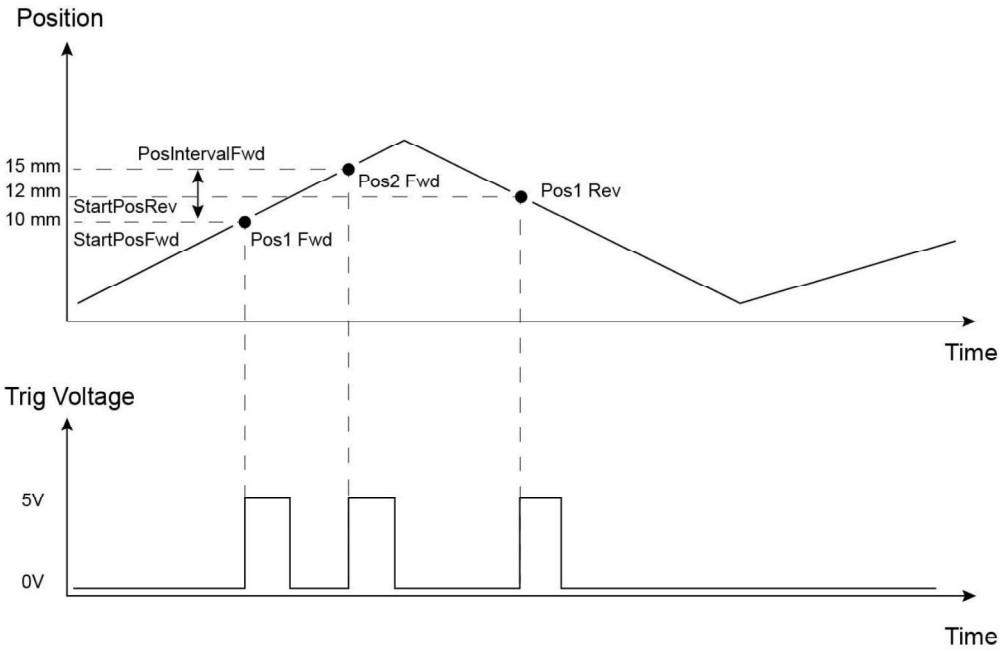
When configured as an output, the TRIG ports can be used as a general purpose digital output, or to indicate motion status or to produce a trigger pulse at configurable positions as follows:

0x0A General purpose logic output (set using the MOD_SET_DIGOUTPUTS message).
0x0B Trigger output active (level) when motor 'in motion'. The output trigger goes high (5V) or low (0V) (as set in the lTrig1Polarity and lTrig2Polarity parameters) when the stage is in motion.
0x0C Trigger output active (level) when motor at 'max velocity'.
0x0D Trigger output active (pulsed) at pre-defined positions moving forward (set using StartPosFwd, IntervalFwd, NumPulsesFwd and PulseWidth parameters in the SetKCubePosTrigParams message). Only one Trigger port at a time can be set to this mode.
0x0E Trigger output active (pulsed) at pre-defined positions moving backwards (set using StartPosRev, IntervalRev, NumPulsesRev and PulseWidth parameters in the SetKCubePosTrigParams message). Only one Trigger port at a time can be set to this mode.
0x0F Trigger output active (pulsed) at pre-defined positions moving forwards and backward. Only one Trigger port at a time can be set to this mode.

Trigger Out Position Steps

In the last three modes described above, the controller outputs a configurable number of pulses, of configurable width, when the actual position of the stage matches the position values configured as the Start Position and Position Interval - see SetKCubePosTrigParams message. These modes allow external equipment to be triggered at exact position values. The position pulses are generated by dedicated hardware, allowing a very low latency of less than 1 usec. The low latency of this triggering mode provides a very precise indication of a position match (assuming a stage velocity of 10 mm/sec, the less than 1 usec latency would in itself only result in a 10 nm position uncertainty, which is normally well below the accuracy limitations of the mechanics.)

Using the last three modes above, position triggering can be configured to be unidirectional (forward or reverse only) or bidirectional (both). In bidirectional mode the forward and reverse pulse sequences can be configured separately. A cycle count setting (set in the SetKCubePosTrigParams message, lNumCycles parameter) allows the uni- or bidirectional position triggering sequence to be repeated a number of times.



Example for a move from 0 to 20 mm and back.

In forward direction: The first trigger pulse occurs at 10 mm (StartPosFwd), the next trigger pulse occurs after another 5 mm (PosIntervalFwd), the stage then moves to 20 mm.

In reverse direction: The next trigger occurs when the stage gets to 12 mm.

Please note that position triggering can only be used on one TRIG port at a time, as there is only one set of position trigger parameters.

The operation of the position triggering mode is described in more detail in the SetKCubePosTriggerParams method.

REQ:
Command structure (6 bytes):

0	1	2	3	4	5
header only					
24	05	Chan Ident	00	d	s

Example: Request the settings for the top panel wheel

TX 24, 05, 01, 00, 50, 01

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

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
25	05	0C	00	d	s	Chan Ident		Trig1Mode		Trig1Polarity	

12	13	14	15	16	17
Data					
Trig2Mode		Trig2Polarity		Reserved	

For structure see SET message above.

MGMSG_MOT_SET_KCUBEPOSTRIGPARAMS 0x0526
MGMSG_MOT_REQ_KCUBEPOSTRIGPARAMS 0x0527
MGMSG_MOT_GET_KCUBEPOSTRIGPARAMS 0x0528

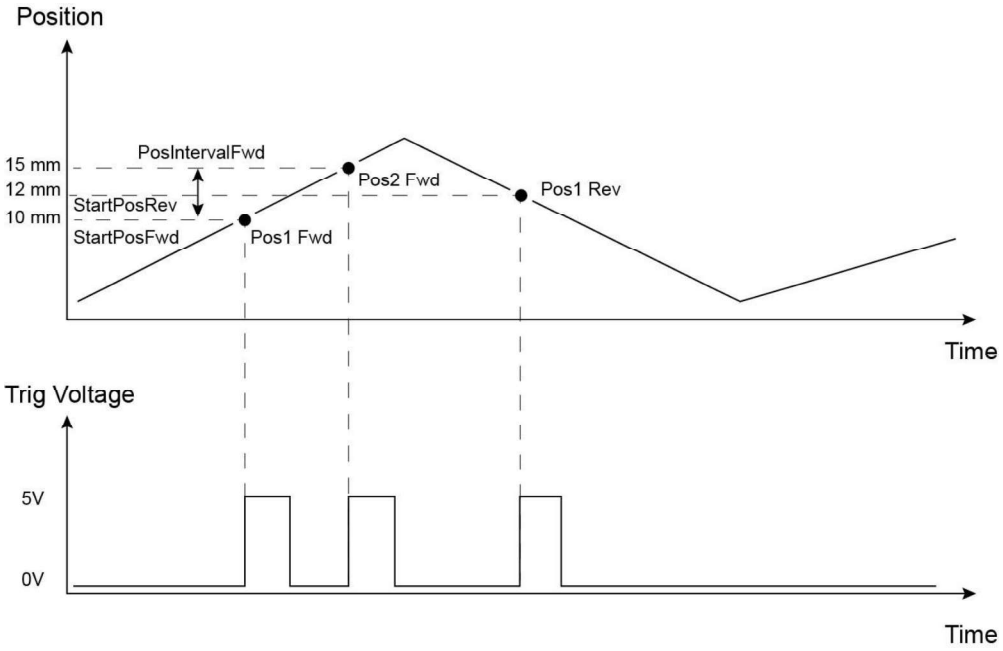
This message is applicable only to KST101, KDC101 and KBD101 units

Function: The K-Cube motor controllers have two bidirectional trigger ports (TRIG1 and TRIG2) that can be set to be used as input or output triggers. This method sets operating parameters used when the triggering mode is set to a trigger out position steps mode by calling the SetKCubeTrigIOConfig message.

As soon as position triggering is selected on either of the TRIG ports, the port will assert the inactive logic state. As the stage moves in its travel range and the actual position matches the position set in the StartPosFwd parameter, the TRIG port will output its active logic state. The active state will be output for the length of time specified by the PulseWidth parameter, then return to its inactive state and schedule the next position trigger point at the "StartPosFwd value plus the value set in the fPosIntervalFwd parameter. Thus when this second position is reached, the TRIG output will be asserted to its active state again. The sequence is repeated the number of times set in the NumPulsesFwd parameter.
When the number of pulses set in the NumPulsesFwd parameter has been generated, the trigger engine will schedule the next position to occur at the position specified in the StartPosRev parameter. The same sequence as the forward direction is now repeated in reverse, except that the PosIntervalRev and NumPulsesRev parameters apply. When the number of pulses has been output, the entire forward-reverse sequence will repeat the number of times specified by NumCycles parameter. This means that the total number of pulses output will be NumCycles x (NumPulsesFwd + NumPulsesRev).

Once the total number of output pulses have been generated, the trigger output will remain inactive.

When a unidirectional sequence is selected, only the forward or reverse part of the sequence will be activated.



Example for a move from 0 to 20 mm and back.

In forward direction: The first trigger pulse occurs at 10 mm (StartPosFwd), the next trigger pulse occurs after another 5 mm (PosIntervalFwd), the stage then moves to 20 mm.

In reverse direction: The next trigger occurs when the stage gets to 12 mm.
Note that the position triggering scheme works on the principle of always triggering at the next scheduled position only, regardless of the actual direction of movement. If, for example, a position trigger sequence is set up with the forward start position at 10 mm, but initially the stage is at 15 mm, the first forward position trigger will occur when the stage is moving in the reverse direction. Likewise, if the stage does not complete all the forward position trigger points, the reverse triggering will not activate at all. For normal operation it is assumed that all trigger points will be reached during the course of the movement.

SET

Command structure (40 bytes)

6 byte header followed by 34 byte data packet.

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
26	05	22	00	d	s	Chan Ident		StartPosFwd			

12	13	14	15	16	17	18	19	20	21	22	23
Data											
IntervalFwd				NumPulsesFwd				StartPosRev			

24	25	26	27	28	29	30	31	32	33	34	35
Data											
IntervalRev				NumPulsesRev				PulseWidth			

36	37	38	39
Data			
NumCycles			

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
StartPosFwd -	When moving forward, this is the stage position [in position counts - encoder counts or microsteps] to start the triggering sequence.	long
IntervalFwd	When moving forward, this is the interval [in position counts - encoder counts or microsteps] at which to output the trigger pulses.	long
NumPulsesFwd	Number of output pulses during a forward move.	long
StartPosRev -	When moving backwards, this is the stage position [in position counts - encoder counts or microsteps] to start the triggering sequence.	long
IntervalRev	When moving backwards, this is the interval [in position counts - encoder counts or microsteps] at which to output the trigger pulses.	long
NumPulsesRev	Number of output pulses during a backwards move.	long
PulseWidth	Trigger output pulse width (from 1 µs to 1000000 µs).	long
NumCycles	Number of forward/reverse move cycles.	long

REQ:

Command structure (6 bytes):

0	1	2	3	4	5
header only					
27	05	Chan Ident	00	d	s

Example: Request the settings for the top panel wheel

TX 27, 05, 01, 00, 50, 01

GET:
Response structure (40 bytes):
6 byte header followed by 34 byte data packet.

0	1	2	3	4	5	6	7		8	9	10	11
header							*Data*					
28	05	22	00	d	s	Chan Ident			StartPosFwd			

12	13	14	15	16	17	18	19	20	21	22	23
Data											
IntervalFwd				NumPulsesFwd				StartPosRev			

24	25	26	27	28	29	30	31	32	33	34	35
Data											
IntervalRev				NumPulsesRev				PulseWidth			

36	37	38	39
Data			
IntervalFwd			

For structure see SET message above.

MGMSG_MOT_SET_KCUBEKSTLOOPPARAMS 0x0529
MGMSG_MOT_REQ_KCUBEKSTLOOPPARAMS 0x052A
MGMSG_MOT_GET_KCUBEKSTLOOPPARAMS 0x052B

This message is applicable only to KST101 and BSC20X units

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.

SET:
Command structure (42 bytes)
6 byte header followed by 36 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11	12	13
header						*Data*							
29	05	24	00	d|	s	Chan Ident		LoopMode		Prop		

14	15	16	17	18	19	20	21	22	23	24	25
Data											
Int				Diff				PIDClip			

26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
Data															
PIDTol				EncoderConst				Not Used							

Data Structure:

field	description	format
Chan Ident	The channel being addressed	word
LoopMode	Sets Open or Closed Loop as follows: 1 Open Loop 2 Closed Loop	word
Prop	The proportional gain. Together with the Integral and Differential, these terms determine the system response characteristics and accept values in the range 0 to 16777216.	long
Int	The integral gain. Together with the Proportional and Differential, these terms determine the system response characteristics and accept values in the range 0 to 16777216.	long
Diff	The differential gain. Together with the Proportional and Integral, these terms determine the system response characteristics and accept values in the range 0 to 16777216.	long
PIDClip	The PIDClip parameter is used to cap the value of the PID loop to prevent runaway at the output. It accepts values in the range 0 to 16777216. If set to 0 then the output of the PID loop is ignored.	long
PIDTol	Value below which the output of PID generator is effectively	long

	deemed to be zero to avoid continual cycle about set point	
EncoderConst	This is a conversion factor from Encoder counts to microsteps. If set to 0, then no encoder is fitted to the stage.	DWord

Example: Set the PID parameters as follows:
Loop Mode: Closed Loop
Prop: 20000
Int: 1000
Diff: 100
PIDClip: 100,000
PidTol: 200
EncoderConst: 4292282941 (see note below

TX 29, 05, 24, 00, D0, 01, 01, 00, 02, 00, 20, 4E, 00, 00, E8, 03, 00, 00, 64, 00, 00, 00, 00, E1, F5, 05, C8, 00, 00, 00, C3, F5. 28, 00, 00, 00, 00, 00, 00, 00, 00, 00,

Header: 29, 05, 24, 00, D0, 01: Set_KCubeKSTLoopParams, 36 byte data packet, Generic USB Device.
Chan Ident: 01, 00: Channel 1 (always set to 1 for BSC201)
LoopMode: 02, 00 : Closed Loop
Prop: *20, 4E, 00, 00:* Set the proportional term to 20000
Int: *E8, 03,:* Set the integral term to 1000
Diff: 64, 00,: Set the differential term to 100
PIDClip: 00, E1, F5, 05,: Set the integral limit to 100,000,000
PIDTol: C8, 00, 00, 00
EncoderConstl: C3, F5, 28, 00, : Set the Enccoder Constant to 4292282941.

Note. Calculating the EncoderConst Value
Each stage has a specific constant for converting encoder counts to microsteps. For the LNR50SE stage, this value is 4292282941.

For example
Encoder resolution = 100 nm
Stepper resolution = 409600 microsteps/turn/mm
= 2.44 nm per step
Therefore no. of µsteps per encoder count = 100 nm/2.44 = 40.96.

The chip inside the controller uses 16.16 bit format, where 16 bits represent the integer and 16 bit are for the fraction.

Interger part 40 = Hex28 = 0X0028
Fraction part 0.96/1/65536 = 62914.56 = F5C3
Therefore EncoderConst value = **0028F5C3**

For negative values, we must find the 2s compliment value…
28F5C3 = 0000 0000 0010 1000.1111 0101 1100 0011
2s comp = 1111 1111 1101 0111.0000 1010 0011 1100 + 1
= **FFD7.0A3D**

REQUEST:
Command structure (6 bytes):

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

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

0	1	2	3	4	5	6	7	8	9	10	11	12	13
header						*Data*							
2B	05	24	00	d|	s	Chan Ident		LoopMode		Prop		

14	15	16	17	18	19	20	21	22	23	24	25
Data											
Int				Diff				PIDClip			

26	27	28	29	30	31	32	33	34	35
Data									
PIDTol				EncoderConst				Not Used	

For structure see Set message above.