LEP MAC6000 Command Manual

Version 1885 4/1/2013 10:34:24 AM

PURPOSE - This purpose of the manual is to describe the LEP MAC6000 commands, structure and syntax.

Also see MAC6000 CAN Enumeration Protocol, MAC 6000 Reports

COMMAND GROUPS - Commands are defined in groups by there command number

- 10-39 REPORT COMMANDS - These commands are sent by the modules on certain events. These reports are asynchronous in nature. For the user to enable these reports, the user must enable them with the REPORTS command.
- 40-49 FIRMWARE DOWNLOAD COMMANDS - These commands are used for updating the firmware code.
- 65 ACTION COMMANDS - These commands cause the module to do some sort of action commands. Action commands are unique in that they always make the module busy. The user sending these commands can always assume the module will become busy. The user must then either poll the module for non-busy status or (if reports are enabled) wait for the non-busy report.
- 83 SET_LONG_DATA - These commands set a index base register value with the data passed.
- 84 GET_LONG_DATA - These commands get a index base register value.
- * All other commands are special commands that don't fit in to the above groups.

COMMAND STRUCTURE - The LEP MAC6000 commands are binary in form. The command always starts with a '#'(0x23) and terminate with a carriage return (0x0D). At a mimimun the command requires a '#', Device Number, CommandNumber, ReservedByte, IndexNumber, DataLength and the termination byte(0x0D). Valid device number are 0,1-31 and 32. Device number 1-31 addresses the individual modules and 32 addresses the interface you are communicating through. A device number of zero sends the command to all modules (global command). For more information on how modules are mapped to device numbers (See MAC6000 CAN Enumeration Protocol). The DataLength must be set to the number of data bytes to follow. If no data is present the DataLength must be set to zero. All data is sent LSB first. Command responses are returned with the same structure. The command response returns the same devicenum, command and index number sent but the MSB bit of the command number will be set high.

- Byte 0 - Start byte always '#'(0x23)
- Byte 1 - Device Number 0-32.
- Byte 2 - Command byte, range 1-127. (MSB bit high for response)
- Byte 3 - Reserved byte. Set to zero.
- Byte 4 - Index LSB
- Byte 5 - Index MSB
- Byte 6 - DataLength LSB
- Byte 7 - DataLength MSB
- Byte 8 - N Data Bytes, Optional (LSB First)
- Byte (8+N) - Termination byte '/n' (0xD)
  Examples
  0x23,0x01,0x53,0x00,0x05,0x00,0x04,0x00,0x05,0x00,0x00,0x00,0x0D - Set position of module 1 to 5.
  0x23,0x01,0x54,0x00,0x05,0x00,0x00,0x00,0x0D - Get position of module 1.
  0x23,0x01,0xD4,0x00,0x05,0x00,0x04,0x00,0x05,0x00,0x00,0x00,0x0D - Response to get position of module 1.

CAN ASCII COMMAND - A 'CAN' command has been added to the interface to allow the MAC6000 commands to be sent in ASCII format. The syntax of the command is: CAN DevNum, CMDNum, Index, Signed_Data, CR - Where DevNum is the Device label or number, CMDNum is the command number, Signed_Data is a signed 4 byte data and CR is the terminator byte. All MAC6000 commands have a data length of 4 with the exception of the wrapper command. The wrapper command is used to send ASCII commands through the MAC6000 binary command structure shown above. These wrapper commands cannot be sent with the ASCII CAN command. Commas are optional. The command response data will always be returned as a ASCII signed number. An example to read position on module 1 would be *CAN 1 84 5 0 CR*.

MANUAL LAYOUT - This manual lists commands by the ascending command and index command. Except of command 83-SET_LONG_DATA and 84-GET_LONG_DATA which are listed together.

CMD# CMD_LABEL - Command Description
DataOut: Describes the sent data packet
DataIn: Describes the response data packet
Index: INDEX_NUM INDEX_LABEL
Desc: Detailed Description

NOTE: DataIn and DataOut are referenced to the module sending the command. Not all commands are issued from the interface.

NOTE: Except on Wrapper command if the specified DataOut or DataIn is more than 4 bytes, the first two bytes will be referring to the INDEX and the next 4 will be the DATA.

NOTE: Most SET_LONG_DATA commands show the DataIn field as being empty. That is because unless otherwise noted there is no response for a SET_LONG_DATA command. There are a few exceptions and they are called out in the following descriptions.

NOTE: GET_LONG_DATA and SET_LONG_DATA indices 80-200 are for special modules are are under development.

---

1 WHOS_OUT_THERE - *Who's there command*

DataOut: n/a

DataIn: Spare(1Byte), Node ID (1Byte), Serial Number (4Bytes)

Index:

Desc: This command is sent from the master interface. On receipt of this command the modules should return there node id and serial number. Module who's serial number have not been assigned should return zero for the serial number. Module with a serial number of zero will not be enumerated.
The returned serial number here is the actual serial number times by 10, plus the channel number.

---

4 SET_NODE_ID - *Sets the modules node id*

DataOut: Spare(1Byte), NodeId(1Byte), SerialNumber(4Bytes)

DataIn:

Index:

Desc: Tells the module whose serial number is matched to set or change it's node id. This command is sent from the master interface during enumeration. If the NodeId is zero the module should disable CAN reception on all node except the global node (zero).
Valid Node ID is 0-31.
The returned serial number here is the actual serial number times by 10, plus the channel number.
On Dual Axis modules the channel number refers to the axis. For example if its an X-Y module then X would be channel 0 and Y would be channel 1. So if the module has a SN or 176252 then the serial number for the first axis would be 176250 and the second would be 176251

---

9 WRAPPER_COMMAND - *Used to encapsulate variable length data. (interface*

only)

DataOut:

DataIn:

Index:

Desc: This command is used between the interface and host software to encapsulate variable length data in a MAC6000 command.

Syntax # DevNum CmdNum Spare Index(2Bytes) Length(2Bytes) Data(Variable Length) CR.

The Encapsulated/embedded data can be any valid interface ASCII or Low Level command. Embedded commands must be complete, partial commands not allowed. If the embedded command is a request command, data is returned with the same format. The embedded command length is limited to 2KBytes(preliminary).

Purpose of this command is allow for an event driven communication system. This is accomplished by wrapping all communication to and from the interface and the user in a standard format. There by allowing the interface and user the capabillity to parse regular commands and asynchronous reports simutanuously.

---

10 REPORT_BOOTUP - *Module bootup report*

DataOut: Spare (1Byte), NodeID (1Byte), Serial Number (4Bytes)

DataIn:

Index:

Desc: This command is sent when the module boots up. It sends the same data as the WHO_OUT_THERE response command. This function will allow modules to announce there presence on the CAN bus.
On boot up all modules (except the master interface) must bootup with an node id of 0. Upon receiving this command the interface will enumerate the module if it's not already enumerated. This will allow modules to be added to the CAN bus dynamically.
The returned serial number here is the actual module serial number times by 10, plus the channel number. See command 4 for description of channel number.

---

11 REPORT_ONLINE - *Module online report*

DataOut Mode(2Byte), SerialNumber(4Bytes)

:

DataIn:

Index:

Desc: This command is sent when a module needs to remove or add itself from the CAN bus. For example if a module needs to rewrite flash and can not respond to CAN commands. After the Offline command has been sent the module should send an online command when it's ready to receive CAN commands again.

Mode Values: 0-Offline, 1-OnLine, 2-ShutDown.
Offline should be used when the module will return and wants to remain enumerated (ie keep the same node id). Offline should be followed by an Online.
ShutDown is sent when the module doesn't want to keep it's enumerated node id(eg. It wants to reboot it self). If a module has encounter a critical error and needs to shut down you can also send this command. After a Shut Down the module should send a REPORT_BOOTUP if is wants to become enumerated again.

The returned serial number here is the actual serial number times by 10, plus the channel number.

---

12 REPORT_SHUTDWN - *Shutdown command*

DataOut :

DataIn:

Index:

Desc: This command tells the module to shut down. The command is usally sent by the interface when a power down event has occured. Modules should turn off motors and other power consuming functions to conserve power. And then enter a safe shutdown state. Generally there is 100ms from when this message is sent to when the power no longer usable.

---

15 REPORT_ERROR - *Module error report*

DataOut : Index(2 bytes) = cmd in error if appl else zero,data = error #

DataIn:

Index:

Desc: This command is sent from a module upon an error event. The 4 bytes of error data are universal.

Also see REQUEST_REPORT_ERROR command.
The first 2 bytes will contain the command that is the source of the error if applicable.
The data (4bytes)will contain the error number -- see preamble for values.
The pipe will be the pipe of the command that caused the error if applicable or the universal pipe (0).

---

20 REPORT_STATUS_POS - *Module status report*

DataOut: Status Data 2bytes + 4 byte position

DataIn:

Index:

Desc: This command is sent from a module upon a status change event. The 2 bytes of status data are universal. Last 4 bytes are position. Also see REQUEST_REPORT_STATUS command.
LSB
move_requested:1; // is motor
move_done:1; // set if move is terminated normally
target_reached:1; // set if completed less than target
user_stop:1; // set if move is stopped by user
invalid_param:1; //not run-invalid param or busy
cw_on:1; // endlimit cw on
ccw_on:1; // endlimit ccw on
soft_low:1; // endlimit soft low on
soft_high:1; // endlimit soft high on
stalled:1; // stalled motor
stop_current:1; // set if completed less than target
stop_temp:1; // set if completed less than target
home_found:1; // home was found
stop_cw:1; // stop from cw
stop_ccw:1; // stop from ccw
from_act_queue:1; // requested from queue

---

21 REPORT_STATUS_MTR - *Module status report*

DataOut: Status Data 6 bytes (2bytes syste) 4 bytes (motor)

DataIn:

Index:

Desc: This command is sent from a module upon a status change event. The 6 bytes of status data are universal. Also see REQUEST_REPORT_STATUS command.

SYSTEM: -- in index
0 - Busy - system busy bit
1 - busy_flash - reading/writing flash
2 - boot running - application invalid running on boot
3 - adu_not_prgmd - adu is not fully prgmd
4 - moving_together - motors request simult move
5 - personality_save - using memory saved in flash
6-15 spare

MTR:
0 - mtr_running:1; // is motor
1 - cw:1; // moving in clockwise direction
2 - rampup:1; // is ramping up
3 - ramping:1; // is ramping
4 - direction:1;// which direction
5 - home_search:1;// doing home checking
6 - boost:1; //higher pwr-faster speed
7 - servo_align:1; //servo move to align at end
8 - endlimit search // endlimit search happening
9 - autodetect // detecting encoder at beginning
10 - stable:1; // stability bit at end of move
11 - open_loop_machine_cnt - using hardware to count open loop
12 - exact_encoder - using exact encoder (6058)
13-15 spare
16-19 shutter1-4
20-23 exposure1-4
24 - preSoftHit - the ramp for soft limit being executed
25 - reCalcRamp - need to recalculate ramp variables
26 - spinCmd - moving because of spin command

---

22 REPORT_ASCII_MSG - *Module reports ascii message*

DataOut: ascii string(6 bytes)

DataIn:

Index:

Desc: This command is sent from a module when it would like to communicate an ascii message. A series of these will be sent, with the last one of the series containing a null terminator to mark the end of message. The interface will collect the messages and upon receeving the null terminator will display them in the debug buffer. Maximum string length is 512 bytes. If REPORTS are enabled these reports will also be sent to the com port.

---

23 REPORT_PWM_POS - *Module reports pwm and position*

DataOut: 1 byte empty + PWM 1bytes + 4 byte position

DataIn:

Index:

Desc: This report can be setup using the status configuration setup. The module will send this at intervals all the time, or just during a requested move, or during all moves or just at the end of a run.

---

24 REPORT_DIAGNOSTIC - *Module reports some undefined diagnostic information*

DataOut: undertermined

DataIn:

Index:

Desc: This report can be setup using the status configuration setup. The module will send this at intervals all the time, or just during a requested move, or during all moves or just at the end of a run. It has not been determined yet. Could be for the temperature or current??

---

25 REPORT_SIGNALS_POS - *Module signal and position report*

DataOut: 2 byte signals, 4 bytes position

DataIn:

Index:

Desc: This command is sent from a module when requested or setup as to be transmitted on a timed interval by SET_STATUS_CONFIG. Also see REQUEST_REPORT_STATUS command.

0 - LIMIT_CW - CW limit signal
1 - LIMIT_CCW CCW limit signal
2 - LIMIT_HOME Home limit signal
3 - LIMIT_PRE PRE limit signal
4 - SOFT_LOW -- low software limit hit
5 - SOFT_HIGH - high software limit hit
6 - SYNC_INPUT sync input signal
7 - SYNC_OUTPUT sync output signal,reset next move
8 - SLEW_BUTTON - slew button pushed
9 - 16 spare

---

26 REPORT_QUEUE_POS - *Module queue status and position report*

DataOut: 2 byte signals, 4 bytes position

DataIn:

Index:

Desc: This command is sent from a module when requested or setup as to be transmitted on a timed interval by SET_QUEUE_CONFIG. Also see REQUEST_REPORT_STATUS command.

---

27 REPORT_SETUP_MTR - *Module setup report*

DataOut: Status Data 6 bytes

DataIn:

Index:

Desc: This command is sent from a module upon an status change event. The 6 bytes of status data are universal. Also see REQUEST_REPORT_STATUS command.

```
motoron:1; // is motor power on
servon:1; // server checking on
joyon:1; // joystick enabled
trackon:1; // trackball enabled
encoderon:1;// encoder enabled
rev_direction:1; // motor reversed
home_enable:1;// enable hw inter
endlimit_low:1;// endlimt active low
all_moves:1; // status transmitted at end of all moves
only_req_moves:1;// status transmitted on req. moves
only_end_of_run:1;// status at end of move nly
every_time:1; // status transmitted based on timing
spare:4;
```

---

39 REPORT_RESERVED - *CMD Numbers 10-39 are reserved for reports.*

DataOut:

DataIn:

Index:

Desc:

---

52 GET_ONE_WIRE_INDEX - ***Gets the one wire specified by the index to the value***

DataOut: Index(2Bytes), n/a (4Bytes)

DataIn: Index(2Bytes), One Wire Data (4Bytes)

Index:

Desc: Reads the data from one wire device at index passed.

Also see SET_LONG_DATA/ONE_WIRE_SELECT.

---

53 GET_ONE_WIRE_ADDRESS - ***Gets the one wire specified by the address to the value given***

DataOut: Index(2Bytes), n/a (4Bytes)

DataIn: Index(2Bytes), One Wire Data (4Bytes)

Index:

Desc: Reads the data from one wire device at address passed.

Also see SET_LONG_DATA/ONE_WIRE_SELECT.

---

65 MOTOR_ACTION - ***Action commands***

DataOut: Index(2Bytes)

DataIn:

Index: **---- SEE MOTOR_ACTION COMMAND INDEXS ----**

Desc: Motion action commands are command that cause the module to do an action. When ever an action command is sent to a module the interface will mark that module busy. The module MUST send a REPORT_STATUS_POS report at the end of the action, to tell the interface that the module has become non-busy. This report is required whether or not the action command is successful.

---

65 MOTOR_ACTION - ***Action commands***

DataOut Index(2Bytes), Target Positon(4Bytes)

:

DataIn:

Index: 0 START_MOTOR_TARGET

Desc: Moves motor towards target. Filter wheels should move the filter to the target position. Position is specified in either motor steps or encoder steps depending if the module is running in open loop or closed loop with encoder feedback.

---

65 MOTOR_ACTION - *Action commands*

DataOut : Index(2Bytes), GlobalModuleMask(4Bytes)

DataIn:

Index: 1 START_MOTOR

Desc: Starts a motor move to a previously set target position.

When this command is sent as a global command, the global module mask parameter must be set for those modules which are to act on the command. The global module mask is simply that each bit of the data corresponds to a module. The lsb of the data(bit 0) corresponds to module 1, second bit (bit 1) to module 2 and so on.

For non-global commands this variable is not aplicable.

---

65 MOTOR_ACTION - *Action commands*

DataOut : Index (2 bytes) Data(4Bytes) -- positive is CW, zero CCW

DataIn:

Index: 2 GOTO_ENDLIMIT

Desc: Will start the motor until an endlimit is reached. The Data portion of the command is the speed at which the motor will run until the endlimit is hit.

---

65 MOTOR_ACTION - *Action commands*

DataOut : Index(2Bytes)

DataIn:

Index: 3 INCREMENT_MOTOR

Desc:

---

65 MOTOR_ACTION - ***Action commands***

DataOut: Index(2Bytes), 4 bytes possible increment

DataIn:

Index: 4 INCREMENT_INC

Desc: The data is the amount of encoder or motor counts to increment the motor. If the value is negative it will decrement that number of encoder or motor counts.

---

65 MOTOR_ACTION - ***Action commands***

DataOut: Index(2Bytes)

DataIn:

Index: 5 DECREMENT_MOTOR

Desc:

---

65 MOTOR_ACTION - ***Action commands***

DataOut: Index(2Bytes)

DataIn:

Index: 6 DECREMENT_INC

Desc: Takes data value and decrements the motor that many encoder counts. If the value is negative it will use the absolute value and decrement.

---

65 MOTOR_ACTION - ***Action commands***

DataOut: Index(2Bytes) Speed(4Bytes)

DataIn:

Index: 7 CENTER_HOME

Desc: Finds the center of the motor range of travel. This is done by first rotating the motor in a negative direction till the negative limit is found. Then rotating the motor in the positive direction till the positive limit is found. Once both end limits are found the center position is calculated and the motor is driven to that position.

[Alternate function] If the center pulse is enabled and found during the limit scan, the motor will stop and drive to the location of the center pulse.

WARNING: When in open loop operation, center operation needs to be run at a speed of 45000 or less. Running into endlimits in open loop mode can cause a loss of position.

---

65 MOTOR_ACTION - ***Action commands***

DataOut: Index(2Bytes), Relative Steps(4Bytes)

DataIn:

Index: 9 MOVE_RELATIVE_MICRO

Desc: Moves the motor in motor micro steps (not encoder steps). This value could be negative causing a decrement or positive causing an increment.

---

65 MOTOR_ACTION - ***Action commands***

DataOut: Index(2Bytes), Velocity(4Bytes)

DataIn:

Index: 10 MOVE_VELOCITY

Desc: Moves the motor at a constant velocity. Velocity units are signed steps per second.

---

65 MOTOR_ACTION - ***Action commands***

DataOut: Index(2Bytes), PWM(4Bytes)

DataIn:

Index: 11 MOVE_PWM

Desc: Sets the motor's PWM value at a constant value. Used in DC motors.

---

65 MOTOR_ACTION - ***Action commands***

DataOut: Index(2Bytes), no data

DataIn:

Index: 12 REQUEST_ENCODER_TEST

Desc: Moves the motor 50 steps in one direction, reads encoder position and then does another 500 steps in the same direction. Next it does the same in the other direction to determine open and closed loop variables and to determine if encoder is available. Here it determines whether it is open or closed loop. The 6056/6057 determines whether it uses an external encoder on the other motor connector at this time.
Returns REPORT_STATUS_POS.
Data on request has no bearing on move.

---

65 MOTOR_ACTION - *Action commands*

DataOut: data - Filter Position(4Bytes)

DataIn:

Index: 13 MOVE_FILTER

Desc: Moves the filter to position indicated by the data.
Filter #1 is also home. Filters #1-10.
If the data == ASCII(N) Next Filter
If the data == ASCII(P) Previous Filter
If the data == ASCII(H) Home Filter Wheel
If the data == 0 (0x00) Goto Maximum position
If the data == 127 (0x7F) Ignore move

data = filter #

wheel is selected by can node -- each node is different filter wheel.

If using an ascii CAN command to Move Filter to home one
could send
CAN 17 65 13 72 (where 72 is the decimal value of 'H')
or
CAN 17 65 13 1

---

65 MOTOR_ACTION - *Action commands*

DataOut: no data

DataIn:

Index: 14 MAKE_HOME_OFFSET

Desc: This is equivalent to pushing the filter up switch after the motor has been disabled. A home offset is necessary due to the way the wheel is assembled.

The index mark on the wheel is not necessarily the number 1 filter position. The home offset is the offset from the encoder index mark to the filter #1 postion.

No returned data.

---

65 MOTOR_ACTION - *Action commands*

DataOut: data - Filter Position(4Bytes) high 2bytes aux wheel, low 2 bytes this FW

DataIn:

Index: 15 MOVE_FILTERS_TOGETHER

Desc: Moves each filter to it's position. High 2 bytes is aux wheel, low 2 bytes is this wheel (main).

Moves the filter to position indicated by the data.
Filter #1 is also home. Filters #1-10.
If the data == ASCII(N) Next Filter
If the data == ASCII(P) Previous Filter
If the data == ASCII(H) Home Filter Wheel
If the data == 0 (0x00) Goto Maximum position
If the data == 127 (0x7F) Ignore move

data = filter #

wheel is selected by can node -- each node is different filter wheel.

---

65 MOTOR_ACTION - *Action commands*

DataOut: data - no data

DataIn:

Index: 16 FAKE_SYNCH_SIGNAL

Desc: If the action queue is enabled using
this command will start first command on the action queue much like an external synch signal. If the queue is configured to continue acting on the queue based on the action finishing it will do that or wait for another of these commands or another synch signal. If the action queue is not enabled this will act like a go command if a valid target has been set.

65 MOTOR_ACTION - *Action commands*

DataOut: data - Carousel Position(4Bytes)

DataIn:

Index: 17 MOVE_CAROUSEL

Desc: Moves the carousel to position indicated by the data.
Carousel #0 & #1 is also home. Carousel #1-4.
If the data == ASCII(N) Next Filter
If the data == ASCII(P) Previous Filter
If the data == ASCII(H) Home Filter Wheel
If the data == 0 (0x00) Goto Home position
If the data == 127 (0x7F) Ignore move

if data >= 0x80 (128) it is referring to a move to a specific cassette. Where cassette 1 and 2 are at carousel home position. The data, carousel and cassette positions are as follows:

Data Carousel Cassette
0x80 (128) 1 1
0x81 (129) 1 1
0x82 (130) 1 2
0x83 (131) 2 3
0x84 (134) 2 4
0x85 (135) 3 5
0x86 (136) 3 6
0x87 (137) 4 7
0x88 (138) 4 8

If using an ascii CAN command to Move Carousel to home one could send
CAN 7 65 17 72 (where 72 is the decimal value of 'H')
or
CAN 17 65 17 1

---

65 MOTOR_ACTION - *Set long data command*

DataOut: Index(2Bytes), OneWireDevice(4bytes)

DataIn:

Index: 20 ONE_WIRE_SAVE

Desc: Saves the selected one wire device to the physical one wire device. The device needed to be previously selected from ONE_WIRE_SELECT command.

The one wire device needs 10ms to process this command. Therefore the user should not send commands to the same device without waiting for a response. The device will respond with STATUS_MTR_POS report when done

If dc motor, data will be interpreted as:
MotorA will interpret
Data=0 =>shutter 1
Data=1 =>shutter 2
and data = 100=> shutter 1
data = 101=> shutter2
data = 102=> shutter3
MotorB will interpret
Data =0 =>shutter 2
Data =1 =>shutter 3
and data =100=> shutter1
data =101=> shutter2
data =102=> shutter3

---

65 MOTOR_ACTION - *Set long data command*

DataOut: Index(2Bytes), OneWireDevice(4bytes)

DataIn:

Index: 21 ONE_WIRE_SELECT

Desc: Selects and Reads the selected one wire device to memory.

The one wire device needs time to process this command. Therefore the user should not send this command repetatively to the same device without waiting for busy release. Will respond with STATUS_MTR_POS report when done

If dc motor, data will be interpreted as:
MotorA will interpret
Data=0 =>shutter 1
Data=1 =>shutter 2
and data = 100=> shutter 1
data = 101=> shutter2
data = 102=> shutter3
MotorB will interpret
Data =0 =>shutter 2
Data =1 =>shutter 3
and data =100=> shutter1
data =101=> shutter2
data =102=> shutter3

65 MOTOR_ACTION - *Set long data command*

DataOut: Index(2Bytes), DataSetNum (4Bytes)

DataIn:

Index: 22 PERSONALITY_LOAD

Desc: Almost all values that can be set with a Command to the motor module can be saved in flash. On bootup if there is a "personality" saved it will be loaded instead of the defaults. This is an all or nothing thing. When you save the personality all those settings will come up, with the main exception that if the encoder ratio on bootup doesn't match what is set it will be overridden and if there is an encoder at bootup it will opt to use the encoder.
There can be up to 3 personalities stored in flash.
Setting the personality to zero - tells the module to go back to using the defaults. After setting the personality to zero the user should reboot the machine.

Syntax is:
CAN [devnum] 83 231 [dataset]

Where Devnum is the device number and Dataset is 0-3 and DATASET = 0 RESTORES FACTORY DEFAULT PARAMETERS

---

65 MOTOR_ACTION - *Set long data command*

DataOut: Index(2Bytes), DataSetNum (4Bytes)

DataIn:

Index: 23 PERSONALITY_SAVE

Desc: The personality is essentially all of the motor configuration which includes default motor speeds, joystick speeds, acceleration, servo setup, power, trackball etc. can be saved as well as desired axis id.
The flash can mantain three custom motor personalities.
So it expects the data to be 1,2 or 3 and will save the current personality in that folder.
If data is zero it will reset the motor to default values but will not change anything in the flash. Data greater than 3 will cause an error report.
In order to save encoder ratio CHECK_RATIO_MODE needs to be setup as well or the encoder test at bootup will determine the encoder ratio.

66 STOP_MOTOR - *Halt command*

DataOut: no index Data(4Bytes)

DataIn: none

Index:

Desc: Stop type. Data = 0 default -- hard stop
1 - Hard Stop (immediate stop)
2 - Soft Stop (ramp down)
No Response expected. If there is a move occurring, the stop will end the move which will cause a REPORT_STATUS_POS, if there is no move there is no response.

---

67 QUEUE_START - *Start Queue commands*

DataOut: Index(2Bytes), Data(4Bytes)

DataIn:

Index:

Desc: Starts the queued commands. Software trigger of queue
data = 0 -- start immediate
1 -- start next time motor is not busy
2 -- start on next hardware synch in signal

---

68 QUEUE_STOP - *Stop Queue commands*

DataOut: Index(2Bytes), Data(4Bytes)

DataIn:

Index:

Desc: Stops queue from continuing.
data = 0 -- stops immediate - stops in middle of move.
data = 1 -- stops queue next time motor is not busy.
2 -- stops on next hardware synch in signal(future).

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), Data(4Bytes)

DataIn:

Index: **--- SEE SET INDEXS ---**

Desc:

84 GET_LONG_DATA - ***Read long data command***

DataOut: Index(2Bytes), Data (4Bytes)

DataIn: Index(2Bytes), Data(4Bytes)

Index: **--- SEE GET INDEXS ---**

Desc:

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), DEVICENUM (4Bytes)

DataIn:

Index: 0 CURRENT_DEVICENUM

Desc: Sets the device number that is currently being used. This will not be saved, will not be used as a wanna be device id.

This is used during module enumeration. The interface sends this command to the moudles. The interface also sends this command to the modules during the ASCII MAP command.

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), CURRENT_DEVICENUM (4Bytes)

Index: 0 CURRENT_DEVICENUM

Desc: Gets the device number that is currently being used. This will not be saved, will not be used as a wanna be device id.
Just used in case another module besides the main interface wants to communicate with it (future use).

---

84 GET_LONG_DATA - ***Get long data command***

DataOut Index(2Bytes), n/a(4Bytes)

:

DataIn: Index(2Bytes), Spare(3Bytes),DipSwitch5000(1Byte)

Index: 1 MODULE_LEGACY

Desc: This commands returns MAC5000 legacy information.
Byte1: MAC5000 Dip Switch Settings - Module dependant.
Byte2: Spare - Default to zero
Byte3: Spare - Default to zero
Byte4: Spare - Default to zero

(Internal Note this is a required command. Low level command 105 will request this command.)

---

83 SET_LONG_DATA - *Set long data*

DataOut: Index(2Bytes), Year(2Byte), Month(1Byte), Day(1Byte)

DataIn: Index(2Bytes)

Index: 2 MODULE_DATE

Desc: Year -- full year (2bytes)
Month - (1byte)
Day - (1 byte)

Get/Set the module date information.

84 GET_LONG_DATA - *Get long data*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Year(2Byte), Month(1Byte), Day(1Byte)

Index: 2 MODULE_DATE

Desc: Year -- full year (2bytes)
Month - (1byte)
Day - (1 byte)

Get/Set the module date information.

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), [Type(4Bytes)]

DataIn: Index(2Bytes), Version (4Bytes)

Index: 3 VERSION_NO

Desc: Gets the firmware version of the module. Type is optional, where a type of zero is the main code version and non-zero are sub-firmware versions.

0 - Main Application (same as 10)
10 - APPL_PROGRAM
11 - APPL_DATA
20 - ADU_PROGRAM
30 - PLD_PROGRAM
40 - BOOT_PROGRAM
41 - BOOT_DATA

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), Version (4Bytes)

DataIn:

Index: 3 VERSION_NO

Desc: Only Get Version number (4 bytes) is supported.
If this Set command is called it will be ignored.

---

83 SET_LONG_DATA - *Set long data*

DataOut: Index(2Bytes)), n/a(1Byte), n/a(1Byte), n/a(1Byte), Requested ID(1Byte)

DataIn: Index(2Bytes)

Index: 4 LEP_DEVICE_TYPE

Desc: Used to set the requested module ID. If non-zero this value will override the One Wire Requested ID. Default is zero.

On enumeration the interface reads this value and will configure this module to this module ID if the module ID is available.

To make this command permanent use the PERSONALITY_SAVE or SAVECFG commands and reboot to renumerate.

84 GET_LONG_DATA - *Get long data*

DataOut: Index(2Bytes),n/a(4Bytes)

DataIn: Index(2Bytes), Family(1Byte), Module,(1Byte), Type(1Byte), Requested ID(1Byte)

Index: 4 LEP_DEVICE_TYPE

Desc: This command requests the module family code, module code and type code. Data is retuned MSB to LSB....Family Code is MSB, Requested ID is LSB.

Family Code: 20-MAC2000, 50-MAC5000, 60-MAC6000
Module Code: 50-MotorStepper, 51-MotorDC, 81-FilterDC, etc.

Type Code: LEP Device Types - See below
Set unknown code and future bytes to zero.

Requested ID: This is the LEP device number representing the ascii device letter(X(1),Y(2),Z(3)...) that the module would like to be. Set to zero if the module has no preference. This data is usually read from a one wire device, this allows the peripheral (stage, filter wheel, etc) to dynamically configure the module.

LEP Device Types
"UNDEF", // 0 - Undefined
"EMOT ", // 1 - Stepper Motor - 1.8 degree
"EMOT ", // 2 - Stepper Motor - 0.9 degree
"EMOTM", // 3 - Stepper Motor - Mapper
"SLIDE", // 4 - Stepper Motor - Carousel
"EMOTD", // 5 - DC Motor -
"EMOTS", // 6 - DC Motor - Mapper
"EMOTB", // 7 - DC Motor - Brake
"SLIDE", // 8 - DC Motor - Carousal
"EFILS", // 9 - Filter wheel - Stepper
"EFILS", // 10 - Filter wheel - DC
"PIEZO", // 11 - Piezo driver
"EAFC ", // 12 - Auto Focus - Stepper
"EAFC ", // 13 - Auto Focus - DC
"FFIND", // 14 - Flat Finder - Stepper
"FFIND", // 15 - Flat Finder - DC
"EDAIO", // 16 - Digital Analog input output
"EAFC ", // 17 - Auto Focus
"NOSE ", // 18 - Turret Changer
"HPHCD", // 19 - Photometer
"INT ", // 20 - Interface
"BOOT ", // 21 - Boot Code

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), Encoder # (4Bytes)

DataIn: Index(2Bytes), Position(4Bytes)

Index: 5 MOTOR_POSITION

Desc: If not using an encoder this will return the current open loop position.

If using an encoder the command gets the encoder value, does not alter it. If the data equals zero it will read back the current encoder being used.
If the data > 1 the second encoder will be read, for the 6050 that is the external encoder, for the 6054 it is the other motor's encoder.
If the data == 1 then the regular encoder is read back.
So if the stage is operating open loop the user can still query the encoder.

Filter wheel returns the current filter wheel position.

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), Position(4Bytes)

DataIn:

Index: 5 MOTOR_POSITION

Desc: Will change current encoder setting if there is no requested move going on. If there is a requested move (meaning not a joystick, trackball or servo move) the position will be changed. If there is a non-requested move occuring - the move will stop momentarily and the encoder will be adjusted.

In the case of the single motor boards, the data in a set encoder command will not determine which encoder is being set. This command will set the encoder currently being used. A user can switch between encoders by using ENCODER_MODE.

Reading encoder from get_long_data will not alter the encoder. Get the current motor position. Filter wheel should return the current filter wheel position.

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), Encoder(4Bytes)

DataIn:

Index: 6 MOTOR_ENCODER

Desc: Will change encoder setting. If reading back from get_long_data will not alter the encoder.

---

83 SET_LONG_DATA - ***Set long data command***

DataOut Index(2Bytes), Target Position(4Bytes)

:

DataIn:

Index: 7 MOTOR_TARGET

Desc: Will change target setting. If reading back from get_long_data will not alter the encoder. This value can be changed while a motor is moving as it is not loaded until an action command is received.

If SYNCH_COORDINATE_CONFIG has been previously sent to set up using a target queue, this will put each target on the queue consecutively. The queue is 100 position long, if more than that are sent they will overwrite each other.

84 GET_LONG_DATA - *Get long data command*

DataOut : Index(2Bytes),

DataIn: Index(2Bytes), Target Position(4Bytes)

Index: 7 MOTOR_TARGET

Desc: Get the motor target setting. If the target queue is being used the data will be used as the index into the queue.

---

83 SET_LONG_DATA - *Set long data command*

DataOut : Index(2Bytes), Encoder(4Bytes)

DataIn:

Index: 8 MOTOR_INCREMENT

Desc: Will change increment setting. This is the value intended to be used in conjunction with the MOTOR_ACTION commands INCREMENT_MOTOR and DECREMENT_MOTOR.

84 GET_LONG_DATA - *Get long data command*

DataOut : Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Increment Value(4Bytes)

Index: 8 MOTOR_INCREMENT

Desc: Read the motor increment position.

---

83 SET_LONG_DATA - *Set long data command*

DataOut Index(2Bytes), MotorDisabled(4Bytes)

:

DataIn:

Index: 9 MOTOR_DISABLED

Desc: If data > 0 the motor will be disabled. The motor will not run, there will be no errors generated from motor commands.
Default is motor enabled unless it is a single motor board.
If data == 0 - motor will be reenabled. If a dual motor it will boot up as two and you can disable them individually.

---

83 SET_LONG_DATA - *Set long data command*

DataOut : Index(2Bytes), Center(4Bytes)

DataIn:

Index: 10 CENTER_POSITION

Desc: Will change center . If reading back from get_long_data will not alter the value.

In the normal stepper(6054,6050,6056) this center value can be written to but will not be used for anything. When written to the home signal interrupt becomes disabled.

In the absolute encoder version (6058), the carousel positoins will be written to using this command.
Sending with data of 0 or 1 will save the current position in for carousel position 1 in the ram as well as the one wire. Sending this command with data 2,3,and 4 will likewise save the current position as carousel position 2,3 and 4. Then a subsequent center command (MOTOR_ACTION / CENTER_HOME) will use that center value (or carousel 1 for the 6058) as a target and go to it.

84 GET_LONG_DATA - *Get long data command*

DataOut : Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Center(4Bytes)

Index: 10 CENTER_POSITION

Desc: Read the motor center ot home position.
If the home signal is enabled this will reflect the position saved from a home signal event. If the home signal is not present and a center home command was executed this will reflect the resulting position from that command. If the absolute encoder version is used (6058)
it will read back a value that may have been previously stored (maintained in the one-wire).

---

83 SET_LONG_DATA - ***Set long data***

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn:

Index: 11 MOTOR_6050_Y

Desc: If sent to a 6050 module will make that axis request to be Y. Must save flash and reboot.

84 GET_LONG_DATA - ***Get long data***

DataOut: Index(2Bytes)

DataIn: Index(2Bytes), data(4 Bytes)

Index: 11 MOTOR_6050_Y

Desc: If sent to a 6050 module will make that axis request to be Y. Must save flash and reboot.

---

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Speed(4Bytes)

Index: 12 MOTOR_STARTING_SPEED

Desc: Will get the motor starting speed. If reading back from get_long_data will not alter the value.

Units: pulse per second
Minimum: 100
Maximum: 20000 -- will attempt anything but would stay under
Default: 5000

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), Speed(4Bytes)

DataIn:

Index: 12 MOTOR_STARTING_SPEED

Desc: Will change motor starting speed. If reading back from get_long_data will not alter the value.

Units: pulse per second
Minimum: 100
Maximum: 20000 -- will attempt anything but would stay under
Default: 5000

If the top speed is less than the start speed, the start and stop speed will be set to this top speed setting and it will run at a constant speed.

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), Speed(4Bytes)

DataIn:

Index: 13 MOTOR_TOP_SPEED

Desc: Will change motor top speed. If reading back from get_long_data will not alter the value.
Units: pulse per second

Exception!! If the top speed is less than the start speed the start speed will be changed to match the top speed and the controller will not do a ramp up and will do a constant speed. If the start speed is below top speed ramping will be attempted. Minimum start speed is 100 in order to have ramping. Speeds below this will be constant.

For the Filter Wheel -- The Mac 5000 command 90 [write speed constant] or command 122 [read speed constant] will be translated differently now.
The one byte speed constant will be multiplied by 500 to give speed in steps/sec . If a speed constant of 17 is sent, this can command will be sent to the filter wheel with a data of 8500 steps/sec. The filter wheel will not go to a known max speed for each filter. The max value is around 12,000 steps/sec. Note this max speed is not necessarily reached but is used in making the curve.

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Speed(4Bytes)

Index: 13 MOTOR_TOP_SPEED

Desc: Will request motor top speed. Units: pulse per second

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Speed(4Bytes)

Index: 14 MOTOR_RESOLUTION

Desc: Returns the motor resolution.
When running the motor a user should not ask about the motor power or resolution as the adu is not designed to communicate while running the motor.

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), Resolution(4Bytes)

DataIn:

Index: 14 MOTOR_RESOLUTION

Desc: Will change motor resolution. This value will be sent down to the ADU. It will not be saved in flash unless a SAVECFG is issued. If changed after the controller has booted up the joystick defaults will be altered to accomodate the change. So a change from 10000 Resolution to 40000 resolution will cause the default joystick speeds (high and low) to be 4 times higher. The servo speed and slow speed for moves like the move off the endlimits are adjusted at this time as well as the encoder ratio. If you then save the configuration to flash all these new variables will be set.
When running the motor a user should not ask about the motor power or resolution as the adu is not designed to communicate while running the motor.

If reading back from get_long_data will not alter the value.

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), Power (4Bytes)

DataIn:

Index: 15 MOTOR_POWER_LEVEL

Desc: Will change motor power. This value will be sent down to the ADU. Power Setting 0- off, 1 - brake, 20 - 20%, 40-40%, 60 -60%, 80 -80%, 100- 100%, 120- 120%

If reading back from get_long_data will not alter the value.
When running the motor a user should not ask about the motor power or resolution as the adu is not designed to communicate while running the motor.

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Power(4Bytes)

Index: 15 MOTOR_POWER_LEVEL

Desc: Will report motor power. This request will be sent down to the ADU, so if the ADU is non-responsive this will not return a value. Power Setting 0- off, 1 - brake, 20 - 20%, 40-40%, 60 -60%, 80 -80%, 100- 100%, 120- 120%

If reading back from get_long_data will not alter the value.
When running the motor a user should not ask about the motor power or resolution as the adu is not designed to communicate while running the motor.

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), Position (4Bytes)

DataIn:

Index: 16 SOFT_LIMIT_HIGH_POS

Desc: Will change software maintained end limit. It will be part of the flash personality that can be saved in flash directly (?). If reading back from get_long_data will not alter the value.

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Position (4Bytes)

Index: 16 SOFT_LIMIT_HIGH_POS

Desc: Gets high software maintained end limit.

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Position (4Bytes)

Index: 17 SOFT_LIMIT_LOW_POS

Desc: Gets the low software maintained end limit.

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), Position (4Bytes)

DataIn:

Index: 17 SOFT_LIMIT_LOW_POS

Desc: Will change software maintained end limit. It will be part of the flash personality that can be saved in flash directly (?). If reading back from get_long_data will not alter the value.

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), Speed (4Bytes)

DataIn:

Index: 18 JOYSTICK_NORMAL_SPEED

Desc: Will change joystick normal speed. It will be part of the flash personality that can be saved in flash directly if that command is given. If reading back from get_long_data will not alter the value. Units: pulse per second. The default value is 9800 steps/sec.

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), n/a (4Bytes)

DataIn: Index(2Bytes), Speed (4Bytes)

Index: 18 JOYSTICK_NORMAL_SPEED

Desc: Will get the joystick normal speed. Units: pulse per second. The default value is 9800 steps/sec.

---

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Speed (4Bytes)

Index: 19 JOYSTICK_TOP_SPEED

Desc: Will get joystick top speed. The default value is 70,000 steps/sec. Units: pulse per second

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), Speed (4Bytes)

DataIn:

Index: 19 JOYSTICK_TOP_SPEED

Desc: Will change joystick top speed. It will be part of the flash personality that can be saved in flash directly if that command is given. If reading back from get_long_data will not alter the value. The default value is 70,000 steps/sec. Units: pulse per second

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), DEVICENUM (4Bytes)

Index: 20 REQUESTED_DEVICENUM

Desc: Gets the device number that is normally requested at enumeration as the "want to be" device id.

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), DEVICENUM (4Bytes)

DataIn:

Index: 20 REQUESTED_DEVICENUM

Desc: Sets the device number as a request if there is no onewire.
This is the device id the module will request at enumeration.

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), Value (4Bytes)

DataIn:

Index: 21 OPEN_LOOP_CONSTANT

Desc: Will change OPEN LOOP CONSTANT. If there is an encoder the value will be auto detected at bootup. If reading back from get_long_data will not alter the value.

84 GET_LONG_DATA - *Get long data command*

DataOut Index(2Bytes), n/a (4Bytes)

:

DataIn: Index(2Bytes), Value (4Bytes)

Index: 21 OPEN_LOOP_CONSTANT

Desc: Will report OPEN LOOP CONSTANT. If there is an encoder the value will be auto detected at bootup. If reading back from get_long_data will not alter the value.

---

84 GET_LONG_DATA - *Get long data command*

DataOut : Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Value (4Bytes)

Index: 22 CLOSED_LOOP_CONSTANT

Desc: Will report closed LOOP CONSTANT. If there is an encoder the value will be auto detected at bootup. If reading back from get_long_data will not alter the value.

---

83 SET_LONG_DATA - *Set long data command*

DataOut : Index(2Bytes), Value (4Bytes)

DataIn:

Index: 22 CLOSED_LOOP_CONSTANT

Desc: Will change closed LOOP CONSTANT. If there is an encoder the value will be auto detected at bootup. If reading back from get_long_data will not alter the value.

---

83 SET_LONG_DATA - *Set long data command*

DataOut : Index(2Bytes), SettleTime (4Bytes)

DataIn:

Index: 23 SETTLING_TIME

Desc: Will change motor settling time. This is essentially a delay after finishing a move until the module will release busy. It is in units of milliseconds.

84 GET_LONG_DATA - *Get long data command*

DataOut : Index(2Bytes), N/A(4Bytes)

DataIn: Index(2Bytes), SettleTime (4Bytes)

Index: 23 SETTLING_TIME

Desc: Will read motor settling time. This is essentially a delay to check if motor really got where we think it should. It is usually used in conjunction with servo checking. If reading back from get_long_data will not alter the value.

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), Value (4Bytes)

DataIn:

Index: 24 SERVO_ACTIVATION_DIST

Desc: Will change motor servo activation distance. This is programmed in encoder steps and is the minimum steps the motor should accept being off the desired target before internal software will start moving back to target position. It only works when servo checking function is enabled and there is an encoder. If reading back from get_long_data will not alter the value.

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Value (4Bytes)

Index: 24 SERVO_ACTIVATION_DIST

Desc: Will return the motor servo activation distance. This is programmed in encoder steps and is the minimum steps the motor should accept being off the desired target before internal software will start moving back to target position. It only works when servo checking function is enabled and there is an encoder. If reading back from get_long_data will not alter the value.

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a (4Bytes)

DataIn: Index(2Bytes), Value (4Bytes)

Index: 25 SERVO_SPEED

Desc: Will return the motor servo speed. This is the speed which the motor will attempt to return to target due to servo checking. It only works when servo checking function is enabled. If reading back from get_long_data will not alter the value. If there is a large move it will use the normal operation speed. If just adjusting a few steps, no ramp will use this speed.

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), Value (4Bytes)

DataIn:

Index: 25 SERVO_SPEED

Desc: Will change motor servo speed. This is the speed which the motor will attempt to return to target due to servo checking. It only works when servo checking function is enabled. If reading back from get_long_data will not alter the value. When servo is needed in background (not at end of a move) the motor will use it's normal setting.

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), Trackball speed in pulses/sec(4Bytes)

DataIn:

Index: 27 TRACKBALL_SPEED

Desc: Will change trackball speed. The speed given will be what the trackball runs when at center switch. Switch to left gives 1/5 of that speed and to right gives 2 times that speed. The default track ball speed is 1250 steps/sec.
Units: steps/sec

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Trackball speed in pulses/sec(4Bytes)

Index: 27 TRACKBALL_SPEED

Desc: Will return trackball speed. The speed given will be what the trackball runs when at center switch. Switch to left gives 1/5 of that speed and to right gives 2 times that speed. The default track ball speed is 1250 steps/sec.
Units: steps/sec

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Mask for input(4Bytes)

Index: 28 TRACKJOY_INPUT

Desc: Return which hardware input for trackball from which motor
returns false if out of range
The lowest byte of data D[0] = track 1 input
D[1] = track 2 input
D[2] = joystick 1 input
D[3] = joystick 2 input
If input is set as zero the default will be used.

The Dual Stepper will have trackball 1 & 2 coming in encoder 1 and 2 index as a default. Trackball 3 is the default for the single stepper motor. If not default the number corresponds to the A2D placement of the device

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), Mask for input (4Bytes)

DataIn:

Index: 28 TRACKJOY_INPUT

Desc: Selects which hardware input for trackball for which motor
The lowest byte of data D[0] = track 1 input
D[1] = track 2 input
D[2] = joystick 1 input
D[3] = joystick 2 input
If input is set as zero the default will be used.
Valid input values are 1-3.

The Dual Stepper will have trackball 1 & 2 coming in encoder 1 and 2 index as a default. Trackball 3 is the default for the single stepper motor.

The dual axis module can either act on trackball 1 and 3 or 1 and 2. It can never act on trk 2 and 3 at same time.

To set motor x to use trackball 3 and motor y to use trackball 1 and keep the joystick the same,
send (0x02010103).

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Mask for input(4Bytes)

Index: 28 TRACKJOY_INPUT

Desc: Return which hardware input for trackball from which motor
returns false if out of range

The lowest byte of data D[0] = track 1 input
D[1] = track 2 input
D[2] = joystick 1 input
D[3] = joystick 2 input
If input is set as zero the default will be used.
The valid values for each input is 1-3.

The Dual Stepper will have trackball 1 & 2 coming in encoder 1 and 2 index as a default. Trackball 3 is the default for the single stepper motor.

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), Data(4Bytes)

DataIn:

Index: 29 SYNCH_IMAGE_MASK

Desc: The bits in the data correspond to a filter position. If bit 0 is set when the filter wheel passes or stops at filter 1 the synch out signal will be pulsed. Bit 1 corresponds to filter position 2 etc.

This is intended to be used with MOVE_VELOCITY for image capturing.

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Data(4Bytes)

Index: 29 SYNCH_IMAGE_MASK

Desc: The bits in the data correspond to a filter position. If bit 0 is set when the filter wheel passes or stops at filter 1 the synch out signal will be pulsed. Bit 1 corresponds to filter position 2 etc.

This is intended to be used with MOVE_VELOCITY for image capturing.

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Data(4Bytes)

Index: 30 SYNCH_COMPARE

Desc: Returns the synch compare value. When the encoder count reaches this data * 100 a synch out signal will be triggered and the corresponding status will be set. The synch out signal will correspond to the synch configuration.

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), Data(4Bytes)

DataIn:

Index: 30 SYNCH_COMPARE

Desc: Upon receipt of this command the counter will be reset to zero. When the encoder count reaches this data * 100 a synch out signal will be triggered and the corresponding status will be set. The synch out signal will correspond to the synch configuration.

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Delay(2Bytes), Configuration (4Bytes)

DataIn:

Index: 31 SYNCH_QUEUE_CONFIGURE

Desc: Configure Action Queue --
bit 0: Enable = Action queue is 0=disabled - default
1=enabled
bit 1: Clear Q -
1=when done action taken off queue-default
0=action remains on queue
bit 2: Trigger Externally
0=trigger next action from end of previous
1=trigger from external synch - default
Bit 3: Synch Out/Synch End of Run Active High
1= active high- default
0= active low

Bit 4: Synch In Active High=1 active - default 0=active low
exception: 6050 default is active low

Bit 5: Pulse Synch Out 1= pulse 0 = level
Bit 6: Not Used.

Bit 7: End of Move Signal on Stability Bit
1= use synch end to indicate stability
0= end of move used for end of move -default

Bit 8:Enable external limits in synch connector(focus only)
0- no external limits-default(synch in triggers move)

1= use synch-in signals as limits
Bit 9:Flip External Limits on SynchIn in Focus Motor
Normally = 0 and SynchIn1 is CW limit and SynchIn2 is CCW limit when external limits in synch connector are enabled.
Flip =1 and SynchIn1 =CCW and SynchIn2 = CW

#define ACT_Q_MASK 0x1
#define CLR_Q_MASK 0x2
#define TRG_Q_MASK 0x4
#define SYNCH_OUT_HIGH 0x08
#define SYNCH_IN_HIGH 0x010
#define SYNCH_PULSE 0x020
#define OUT_STABILITY 0x040
#define END_STABILITY 0x080
#define SYNCH_EXT_ENAB 0x0100
#define FLIP_EXT_SYNCH 0x0200

84 GET_LONG_DATA - *Get long data command*

DataOut: Delay(2Bytes), n/a(4Bytes)

DataIn: Delay(2Bytes), Configuration (4Bytes)

Index: 31 SYNCH_QUEUE_CONFIGURE

Desc: Configure Action Queue --
bit 0: Enable = Action queue is 0=disabled - default
1=enabled
bit 1: Clear Q -
1=when done action taken off queue-default
0=action remains on queue
bit 2: Trigger Externally
0=trigger next action from end of previous
1=trigger from external synch - default
Bit 3: Synch Out/Synch End of Run Active High
1= active high- default
0= active low

Bit 4: Synch In Active High=1 active - default 0=active low
exception: 6050 default is active low

Bit 5: Pulse Synch Out 1= pulse 0 = level - default
Bit 6: Not Used.
Bit 7: End of Move Signal on Stability Bit
1= use synch end to indicate stability
0= end of move used for end of move -default

Bit 8:Enable external limits in synch connector(focus only)
0- no external limits-default(synch in triggers move)

1= use synch-in signals as limits
Bit 9:Flip External Limits on SynchIn in Focus Motor
Normally = 0 and SynchIn1 is CW limit and SynchIn2 is CCW limit when external limits in synch connector are enabled.
Flip =1 and SynchIn1 =CCW and SynchIn2 = CW

```
#define ACT_Q_MASK 0x1
#define CLR_Q_MASK 0x2
#define TRG_Q_MASK 0x4
#define SYNCH_OUT_HIGH 0x08
#define SYNCH_IN_HIGH 0x010
#define SYNCH_PULSE 0x020
#define OUT_STABILITY 0x040
#define END_STABILITY 0x080
#define SYNCH_EXT_ENAB 0x0100
#define FLIP_EXT_SYNCH 0x0200
```

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Speed(4Bytes)

Index: 32 MOTOR_BOOST_SPEED

Desc: Will get the motor boosting speed. If zero boost is always on (not recommended), if -1 boost is never on.
otherwise is some speed default 10000 pls/sec
reading back from get_long_data will not alter the value.
Units: pulse per second

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), Speed(4Bytes)

DataIn:

Index: 32 MOTOR_BOOST_SPEED

Desc: Will change motor boosting speed. If zero boost is always on (not recommended), if -1 boost is never on.
otherwise set to some speed default 10000 pls/sec
reading back from get_long_data will not alter the value.
Units: pulse per second

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), Report Selected(1Byte) Not Used (2Byte)Enabled (1 byte)

DataIn:

Index: 33 CONFIG_REPORT_ENABLE

Desc: The report selected is an actual report cmd number (ie 20 for REPORT_STATUS_POS). If the enabled byte is 0 it is no longer sent on the timed interval (The time interval is set using CONFIG_RPT_INTERVAL). If enabled is set to 1 it will be sent at the time interval if there is a change from the last time it was sent.
The time interval is in milliseconds. If there was no change in the report value the report will not be sent to avoid overloading the system with useless commands.

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Report Selected(1Byte) Not Used (2Byte)Enabled (1 byte)

Index: 33 CONFIG_REPORT_ENABLE

Desc: The report selected is an actual report cmd number (ie 20 for REPORT_STATUS_POS). If the enabled byte is 0 it is no longer sent on the timed interval (Interval is set using CONFIG_RPT_INTERVAL). If enabled is set to 1 it will be sent at the time interval if there is a change from the last time it was sent.
This command needs to be sent for each report that wish to inquire whether its enabled.

---

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), ReportNumber (4Bytes)

DataIn:

Index: 34 REQUEST_REPORT

Desc: Tells the module to send report. The report will be sent with msb high to indicate a response. These reports will be used for things like filter status that are called often and won't be looked at as reports to be sent anywhere else but just as a one time response to a request. The report to send is passed as the data variable. e.g. REPORT_ERROR, REPORT_STATUS_MTR, REPORT_STATUS_POS, REPORT_STATUS_FW, REPORT_STATUS_DAIO

NOTE: Not available on all versions of firmware. Use SET_LONG_DATA

REQUEST_REPORT instead.

---

83 SET_LONG_DATA ***- Set long data command***

DataOut: Index(2Bytes), ReportNumber (4Bytes)

DataIn:

Index: 34 REQUEST_REPORT

Desc: Tells the module to send report. The report to send is passed as the data variable. e.g. REPORT_ERROR, REPORT_STATUS_MTR, REPORT_STATUS_POS, REPORT_STATUS_FW, REPORT_STATUS_DAIO

Asynchronous reports -- go to pipe 0
Requested go to pipe it came from.

---

83 SET_LONG_DATA ***- Set long data command***

DataOut: Report Selected(1Byte) Which Two Bytes(1byte) Mask(2Byte)

DataIn:

Index: 35 CONFIG_REPORT_MASK

Desc: Sets the mask for the selected report. Each Report sends 6 bytes. The mask will be ANDed with the selected two bytes.

If Which = 0 -- the index of the report will be masked,
if Which = 1 -- the high two bytes of the report will be masked
if which = 2 -- the low two bytes of the report will be masked.

Ex:
If Index is represented as IIII, and the data is represented as AAAABBBB and the mask is represented as MMMM then:

If which=0 the value looked at would be IIII & MMMM
if which=1 the value looked at would be AAAA & MMMM
if which=2 the value looked at would be BBBB & MMMM

If the status has changed in the value looke at from the last status it will be transmitted at the specified time interval or immediately if it is an automatic report.

Automatic reports include -- REPORT_ERROR, and any hardware change.

83 SET_LONG_DATA - *Set long data command*

DataOut: Index (2Bytes), Time Interval (2Bytes)

DataIn:

Index: 36 CONFIG_RPT_INTERVAL

Desc: Sets the interval the report status is sent. Units 1ms, default value = ???ms. A value of zero tells the system to not send the report unsollicited. The end of move status_position report will still occur.

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a (4bytes)

DataIn: Index(2Bytes), OneWireDevice(4bytes)

Index: 37 ONE_WIRE_PRESENT

Desc: Does not read the one wire.
It will returns 0 if the one wire device is not present, as determined by a previous read, otherwise 1 if present.

If dc motor data will be interpreted as:
MotorA will interpret
Data=0 =>shutter 1
Data=1 =>shutter 2
and data = 100=> shutter 1
data = 101=> shutter2
data = 102=> shutter3
MotorB will interpret
Data =0 =>shutter 2
Data =1 =>shutter 3
and data =100=> shutter1
data =101=> shutter2
data =102=> shutter3

If stepper motor data is ignored

---

83 SET_LONG_DATA - ***Set stability threshold***

DataOut: Index(2Bytes), Not Used (2Bytes), Time Interval (2Bytes)

DataIn:

Index: 38 STABILITY_INTERVAL

Desc: Sets the interval in microseconds that the system will look for stability. If the encoder is within the pre-determined threshold after sampling at this time interval the system will determine it is stable and set a stable bit and send the REPORT_STATUS_MTR which contains the stability bit. If set to zero it will disable the stability check. The actual check is done in increments of 50us so if the interval is set to 120 microseconds it will actually be done every 100 microseconds.

Currently when the system is determined stable the synchOut pin for that motor will be pulsed for 200us. See the SYNCH_QUEUE_CONFIGURE Command for changing those settings.

84 GET_LONG_DATA - ***Get stability threshold***

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Not Used (2Bytes), Time Interval (2Bytes)

Index: 38 STABILITY_INTERVAL

Desc: Gets the maximum time in microseconds that the system will look for stability. If zero will look until next move.

---

83 SET_LONG_DATA - ***Set stability threshold***

DataOut: Index(2Bytes), Not Used(2Bytes), Time Interval (2Bytes)

DataIn:

Index: 39 STABILITY_THRESHOLD

Desc: Sets the threshold in encoder steps that will determine stability at the end of a run. The difference in encoder values between stability intervals needs to be less than this threshold for the system to determine it is stable. Upon determining it is stable it will send a REPORT_STATUS_MTR which contains the stability bit. Currently it defaults to also pulsing the SynchOut pin for that motor for 200us. These settings can be changed see SYNCH_QUEUE_CONFIGURE command.

84 GET_LONG_DATA - ***Get stability threshold***

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Not Used (2Bytes), Time Interval (2Bytes)

Index: 39 STABILITY_THRESHOLD

Desc: Sets the threshold in microsteps that will determine stability of ringing at the end of a run. When this is zero the stablity bit is not enabled.

---

84 GET_LONG_DATA - *Get stability time interval*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), NotUsed(2Bytes), Time Interval (2Bytes)

Index: 40 STABILITY_TIME

Desc: Gets the last time it took to become stable in microseconds (after servo and settling).

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes),

DataIn: Index(2Bytes), OneWireDevice(4bytes)

Index: 41 PERSONALITY_PRESENT

Desc: Does not read the flash data.
Returns > 0 if a personality is in use.
The value is the number of the personality (1,2 or 3)
If the user wants to read the personality see
RD_PGM_START.

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), Home Offset(4Bytes)

DataIn:

Index: 50 HOME_OFFSET

Desc: Saves the data position as the home offset.

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), Home Offset (4 bytes)

DataIn:

Index: 50 HOME_OFFSET

Desc: Gets the current home offset.

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), BaudRate(4Bytes)

Index: 60 BAUD_RATE

Desc: Gets the baud rate.
Note: This command is only valid if sent to the interface module.

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), Baud Rate (4Bytes)

DataIn:

Index: 60 BAUD_RATE

Desc: Sets the baud rate -- FOR INTERFACE ONLY.

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), busy(4Bytes)

Index: 63 MODULE_BUSY

Desc: Gets the module busy status of all the modules. Bit0=interface, bit1=X, bit2=Y ....bit32.
This command is only available on the interface.

Not Busy=0
Busy = 1

Note: A module which is not installed will report as BUSY

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), ModulePresent(4Bytes)

Index: 64 MODULE_PRESENT

Desc: Gets the module present status of all the modules. Bit0=interface, bit1=X, bit2=Y ....bit32.
Bit High when present, else low.
This command is only available on the interface.

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes)

DataIn: Index(2Bytes)

Index: 65 EIGHT_VOLTS

Desc: Reads the voltage on the "8V" power bus that supports step motor low speed operation. Typically closer to 7.0 Volts. The Value given in mVolts .
This command is only available on the single stepper (6050 or 6056).

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), Joystick Acceleration (2Bytes)

DataIn:

Index: 70 JOYSTICK_ACCELERATION

Desc: Sets the joystick acceleration. This is also the acceleration used in the spin commands.

The default value is 71400 steps/sec ^2.

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), NotUsed(2Bytes), Joystick Acceleration (2Bytes)

Index: 70 JOYSTICK_ACCELERATION

Desc: Get the joystick acceleration.

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), Target Position(4Bytes)

DataIn:

Index: 77 MOTOR_TARGET_OFFSET

Desc: Will add to target offset queue. The target offsets are relative moves done after each target move if setup properly. This queue can be changed while a motor is moving as it is not loaded until a synch signal or fake synch command is recieved.

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), queue index(4Bytes)

DataIn: Index(2Bytes), Target Position(4Bytes)

Index: 77 MOTOR_TARGET_OFFSET

Desc: Will retrieve target offset from queue, the data will be used as the index into the queue. The target offsets are relative moves done after each target move if setup properly. This queue can be changed while a motor is moving as it is not loaded until a synch signal or fake synch command is recieved.

---

84 GET_LONG_DATA - *Set long data command*

DataOut: Delay(2Bytes), Configuration (4Bytes)

DataIn:

Index: 80 SYNCH_COORDINATE_CONFIGURE

Desc: Bit 0: Coordinate moves (1 start / 1 stop)
Bit 1: Use target queue with Synch Signal/Synch Command
Bit 2: Use offset queue with Synch Signal/Synch Command

When bit 0 is set the one signal will cause the move of both motors and the end of move 1 will signal both moves being done.

When bit 1 is set the target queue will be cycled through for targets when there is an active synch signal or command.

When bit 2 is set after each target the offsets will be cycled through.

Every time this command is sent all previously set targets and offsets are ignored. This should be set before sending the targets and offsets for the queues.

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Delay(2Bytes), Configuration (4Bytes)

DataIn:

Index: 80 SYNCH_COORDINATE_CONFIGURE

Desc: Bit 0: Coordinate moves (1 start / 1 stop)
Bit 1: Use target queue with Synch Signal/Synch Command
Bit 2: Use offset queue with Synch Signal/Synch Command

When bit 0 is set the one signal will cause the move of both motors and the end of move 1 will signal both moves being done.

When bit 1 is set the target queue will be cycled through for targets when there is an active synch signal or command.

When bit 2 is set after each target the offsets will be cycled through.

Every time this command is sent all previously set targets and offsets are ignored. This should be set before sending the targets and offsets for the queues.

---

84 GET_LONG_DATA - ***Get long data command***

DataOut:

DataIn:

Index: 81 SYNCH_PULSES_IN_CNT

Desc: Keeps count of synch pulses in for that module. Count will be reset upon bootup and recieving the fake synch signal.

---

84 GET_LONG_DATA - ***Get long data command***

DataOut:

DataIn:

Index: 82 EOM_PULSES_OUT_CNT

Desc: Keeps count of end of move pulses out for that module. Count will be reset upon bootup and recieving the fake synch signal.

---

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes)

DataIn: Index(2Bytes), EncoderMode (4Bytes)

Index: 100 ENDAT_ENCODER_ID

Desc: Get the encode ID.

---

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes)

DataIn: Index(2Bytes), EncoderMode (4Bytes)

Index: 101 ENDAT_ENCODER_OPEN_RATIO

Desc: Get the endat encoder open loop ratio

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes)

DataIn: Index(2Bytes), EncoderMode (4Bytes)

Index: 101 ENDAT_ENCODER_OPEN_RATIO

Desc: Sets the endat encoder open loop ratio

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes)

DataIn: Index(2Bytes), EncoderMode (4Bytes)

Index: 102 ENDAT_ENCODER_CLOSED_RATIO

Desc: Sets the endat encoder closed loop ratio

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes)

DataIn: Index(2Bytes), EncoderMode (4Bytes)

Index: 102 ENDAT_ENCODER_CLOSED_RATIO

Desc: Get the endat encoder closed loop ratio

---

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes)

DataIn: Index(2Bytes), EncoderMode (4Bytes)

Index: 103 ENDAT_ENCODER_RESOLUTION

Desc: Get the endat encoder resolution

---

83 SET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes)

DataIn: Index(2Bytes), EncoderMode (4Bytes)

Index: 103 ENDAT_ENCODER_RESOLUTION

Desc: Sets the endat encoder resolution

---

83 SET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes)

DataIn: Index(2Bytes), EncoderMode (4Bytes)

Index: 104 ENDAT_ENCODER_DATASIZE

Desc: Sets the endat encoder data size in bits

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes)

DataIn: Index(2Bytes), EncoderMode (4Bytes)

Index: 104 ENDAT_ENCODER_DATASIZE

Desc: Get the endat encoder data size in bits

---

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes)

DataIn: Index(2Bytes), EncoderMode (4Bytes)

Index: 104 ENDAT_ENCODER_DATASIZE

Desc: Get the endat encoder data size in bits

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes)

DataIn: Index(2Bytes), EncoderMode (4Bytes)

Index: 105 ENDAT_ENCODER_OFFSET

Desc: Get the endat encoder offset. Each encoder reads a range that does not include zero. This value will offset all encoder values.

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes)

DataIn: Index(2Bytes), EncoderMode (4Bytes)

Index: 105 ENDAT_ENCODER_OFFSET

Desc: Sets the endat encoder offset. Each encoder reads a range that does not include zero. This value will offset all encoder values. This value will be written into the endat memory to be retrieved at subsequent power ups. If the data value is zero the current absolute position will be the
new offset, making the current position 0. If the data is 2, the offset will be set to zero so the absolute position is the current position. All other data values will set the offset to that data value.

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes)

DataIn: Index(2Bytes), EncoderMode (4Bytes)

Index: 106 ENDAT_MEMORY

Desc: The endat has lots of information in it's memory, some useful to us some not. This will allow the user to read all of it, just give index (0-47).
See Heidenhain documentation for full description here are some of the more useful memory locations:

Addr / Data
9 / Memory location for oem parameters

10 \
11 /memory location for compensation values
12 \
13 Number of clock pulses in data (22)
14 type of encoder
15 /signal periods per revolution for incrementa
16 \
17 distinguishable revolutions (for multiturn encoders)

20 / resolution - measuring step or steps per rev
21 \
22 /datum shift of the encoder mfg
23 \
24 /
25 < ID number
26 \
27 /
28 < Serial number
29 \
30 direction of rotation

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), MaxTemp(4Bytes)

DataIn: Index(2Bytes)

Index: 120 MAX_CASSETTE_CAPACITY

Desc: Sets the maximum expected cassette capacity (wafer or slide). When scanning for the cassette map this will be used to determine size of array and division of scan field for the existance of a wafer or slide. Default is 25.

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), MaxTemp(4Bytes)

DataIn: Index(2Bytes)

Index: 120 MAX_CASSETTE_CAPACITY

Desc: Gets the maximum expected cassette capacity (wafer or slide). When scanning for the cassette map this will be used to determine size of array and division of scan field for the existance of a wafer or slide. Default is 25.

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), MaxTemp(4Bytes)

DataIn: Index(2Bytes)

Index: 121 FIRST_POSITION_CASSETTE

Desc: Sets the position for the bottom of the wafer or slide occupying the first slot in the cassette. Usually this can be done by using the joystick to move from the bottom of the first slot until the laser makes contact. If sent with the data equal to zero the current position will be recorded. Default is the value of the start scan position.

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), MaxTemp(4Bytes)

DataIn: Index(2Bytes)

Index: 121 FIRST_POSITION_CASSETTE

Desc: Gets the position for the bottom of the wafer or slide occupying the first slot in the cassette. Usually this can be done by using the joystick to move from the bottom of the first slot until the laser makes contact. If sent with the data equal to zero the current position will be recorded. Default is the value of the start scan position.

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), MaxTemp(4Bytes)

DataIn: Index(2Bytes)

Index: 122 LAST_POSITION_CASSETTE

Desc: Sets the position for the top of the wafer or slide occupying the last slot in the cassette. Usually this can be done by using the joystick to move from the bottom of the first slot until the laser makes contact. If sent with the data equal to zero the current position will be recorded. Default is the value of the end scan position.

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), MaxTemp(4Bytes)

DataIn: Index(2Bytes)

Index: 123 START_SCAN_POSITION

Desc: Sets the position for the beginning of the scanning. This should be a position below the first slot position.

It is essentially when the laser goes on.

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), MaxTemp(4Bytes)

DataIn: Index(2Bytes)

Index: 123 START_SCAN_POSITION

Desc: Gets the position for the beginning of the scanning. This should be a position below the first slot position.
It is essentially when the laser goes on.

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), MaxTemp(4Bytes)

DataIn: Index(2Bytes)

Index: 124 END_SCAN_POSITION

Desc: Gets the position for the end of the scanning. This should be a position above the last slot.
It is essentially when the laser goes off.

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), MaxTemp(4Bytes)

DataIn: Index(2Bytes)

Index: 124 END_SCAN_POSITION

Desc: Sets the position for the end of the scanning. This should be a position above the last slot position.
It is essentially when the laser goes off.

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), MaxTemp(4Bytes)

DataIn: Index(2Bytes)

Index: 125 LASER_BEAM_FUNCTION

Desc: Sets whether the laser beam is on or off. This is just used for diagnostic or setup purposes and would not be used in conjunction with a scan command.

0 = off
1 = on

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), MaxTemp(4Bytes)

DataIn: Index(2Bytes)

Index: 125 LASER_BEAM_FUNCTION

Desc: Gets whether the laser beam is on or off. This is just used for diagnostic or setup purposes and would not be used in conjunction with a scan command.
0 = off
1 = on

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), MaxTemp(4Bytes)

DataIn: Index(2Bytes)

Index: 126 LASER_STATUS

Desc: Gets the laser status.
Bit 7 = CCW Limit - active high means on limit
Bit 6 = CW Limit
Bit 5 = not used
Bit 4 = not used
Bit 3 = Laser Power2 - 1= laser on
Bit 2 = Mapper Sense 2 1 = slide detected
Bit 1 = Laser Power 1
Bit 0 = Mapper Sense 1

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), MaxTemp(4Bytes)

DataIn: Index(2Bytes)

Index: 127 SINGLE_SCAN_MAP1

Desc: Gets the map returned on a single scan on laser #1.

if data == 0
Can 3 84 127 0 - returns 4 bytes relating to bottom 16 slots (0-15)
if data == 1
Can 3 84 127 1 - returns 4 bytes relating to next 16 slots (16-31) usually max is 25

if data == 2
Can 3 84 127 2 - returns 4 bytes relating to the top 16 slots (32-47) usually max is 25

MSB LSB Description
0 0 Slot Not Occupied
0 1 Slot Occupied
1 0 More than One Occupant
1 1 Tilted Occupant

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), MaxTemp(4Bytes)

DataIn: Index(2Bytes)

Index: 128 SINGLE_SCAN_MAP2

Desc: Gets the map returned on a single scan on laser #2.

if data == 0
Can 3 84 128 0 - returns 4 bytes relating to bottom 16 slots (0-15)
if data == 1
Can 3 84 128 1 - returns 4 bytes relating to next 16 slots (16-31) usually max is 25
if data == 2
Can 3 84 128 2 - returns 4 bytes relating to the top 16 slots (32-47) usually max is 25

MSB LSB Description
0 0 Slot Not Occupied
0 1 Slot Occupied
1 0 More than One Occupant
1 1 Tilted Occupant

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), MaxTemp(4Bytes)

DataIn: Index(2Bytes)

Index: 129 DOUBLE_SCAN_MAP1

Desc: Gets the map returned on a double scan on laser #1.

if data == 0
Can 3 84 129 0 - returns 4 bytes relating to bottom 16 slots (0-15)

if data == 1
Can 3 84 129 1 - returns 4 bytes relating to next 16 slots (16-31) usually max is 25
if data == 2
Can 3 84 129 2 - returns 4 bytes relating to the top 16 slots (32-47) usually max is 25

MSB LSB Description
0 0 Slot Not Occupied
0 1 Slot Occupied
1 0 More than One Occupant
1 1 Tilted Occupant

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), MaxTemp(4Bytes)

DataIn: Index(2Bytes)

Index: 130 DOUBLE_SCAN_MAP2

Desc: Gets the map returned on a double scan on laser #2.

if data == 0
Can 3 84 130 0 - returns 4 bytes relating to bottom 16 slots (0-15)
if data == 1
Can 3 84 130 1 - returns 4 bytes relating to next 16 slots (16-31) usually max is 25
if data == 2
Can 3 84 130 2 - returns 4 bytes relating to the top 16 slots (32-47) usually max is 25

MSB LSB Description
0 0 Slot Not Occupied
0 1 Slot Occupied
1 0 More than One Occupant
1 1 Tilted Occupant

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Boolean - presence 1, not 0

DataIn: Index(2Bytes)

Index: 131 ASK_CASSETTE

Desc: Returns 1 for presence of a slide/ 0 if none.

---

84 GET_LONG_DATA - ***Read long data command***

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), maxRange(4Bytes)

Index: 200 MAX_RANGE

Desc: Returns the maximum range of a device. Return zero if unknown. Motor driver should return the available range. Filter wheels should return their max number of filter positions.

---

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Current(4Bytes)

Index: 201 MOTOR_CURRENT

Desc: Gets the motor current. Units mA.

---

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), PWM(4Bytes)

Index: 202 MOTOR_PWM

Desc: Gets the motor PWM value.

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), Sample Time(4Bytes)

DataIn:

Index: 203 MOTOR_SAMPLE_TIME

Desc: Sets the motor sample period. Units are in micro-seconds. Used to set the DC motor DSP sample time.

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Sample Time(4Bytes)

Index: 203 MOTOR_SAMPLE_TIME

Desc: Gets the motor sample period. Units are in micro-seconds.
Used to set the DC motor DSP sample time.

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), Joystick Deflection (4Bytes)

DataIn:

Index: 206 JOYSTICK_DEFLECTION

Desc: When setting the deflection for virtual joystick -- it expects the data to be in 8 bit form.

Warning: Setting the deflection value forces the whole controller into simulation mode. No more a2d conversions will be done until this command is sent with a value of 0x80 (128) or a halt is issued. The 8 bit deflection will make values near 255 act as a full deflection in positive direction and zero a full deflection in negative direction. If you don't send both modules of a dual module controller the present a2d values will be used.

When requesting deflection (CMD 84 index = 206)
when data == 0 an 8 bit version of the deflection is returned (0-255) where zero deflection should be around 128 and full deflection in positive direction should be close to 255 and full deflection in the other direction should be close to zero. This takes into account subtracting the home as saved from startup.

when data == 1 a 16 bit version of the raw deflection is returned with zero around 0x7fff. This value is based on 3.3 volts.
when data == 2 a 16 bit version of the home is returned.
This is the raw value that is subtracted from the deflection and then divided by assumed max deflection for percent of deflection.

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), JoyStick Deflection (4Bytes)

Index: 206 JOYSTICK_DEFLECTION

Desc: When requesting deflection (CMD 84 index = 206)
when data == 0 an 8 bit version of the deflection is returned (0-255) where

zero deflection should be around 128 and full deflection in positive direction should be close to 255 and full deflection in the other direction should be close to zero. This takes into account subtracting the home as saved from startup.

when data == 1 a 16 bit version of the raw deflection is returned with zero around 0x7fff. This value is based on 3.3 volts.
when data == 2 a 16 bit version of the home is returned.
This is the raw value that is subtracted from the deflection and then divided by assumed max deflection for percent of deflection.

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), Joystick Virtual (4Bytes)

DataIn:

Index: 207 JOYSTICK_VIRTUAL

Desc: Will change joystick perceived deflection. Allows the user to software simulate the joystick. The physical joystick should be turned off prior to using this command.

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Joystick Virtual (4Bytes)

Index: 207 JOYSTICK_VIRTUAL

Desc: Gets the virtual joystick value.

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), Position(4Bytes)

DataIn:

Index: 208 MOTOR_SECOND_POSITION

Desc: Sets the second motor position.

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Position(4Bytes)

Index: 208 MOTOR_SECOND_POSITION

Desc: Reads the second motor position.

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes)Front Panel Mode (4Bytes)

DataIn:

Index: 209 FRONT_PNL_MODE

Desc: Set the front panel switches mode, 0=off, 1=On.

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Front Panel Mode (4Bytes)

Index: 209 FRONT_PNL_MODE

Desc: Gets the front panel switches mode, 0=off, 1=On.

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), Acceleration(4Bytes)

DataIn:

Index: 210 ACCEL_UP

Desc: Will change acceleration. This doesn't affect acceleration on the joystick or the spin commands. To change those see JOYSTICK_ACCELERATION.
Units: pulse per sec^2.
Maximum acceleration is 6000000
Minimum acceleration is 5600
Default acceleration is 300000
High level accel commands there is no equivalent so the accel command will come back as zero.

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Acceleration(4Bytes)

Index: 210 ACCEL_UP

Desc: Get acceleration.
Units: pulse per sec^2

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), Decceleration(4Bytes)

DataIn:

Index: 211 ACCEL_DWN

Desc: Will change deceleration. This doesn't affect deceleration on the joystick or the spin commands. To change those see JOYSTICK_ACCELERATION.
Units: pulse per sec^2.
Maximum acceleration is 6000000
Minimum acceleration is 5600
Default acceleration is 300000
High level accel commands there is no equivalent so the accel command will come back as zero.

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Deceleration(4Bytes)

Index: 211 ACCEL_DWN

Desc: Will read deceleration. Units: pulse per sec^2

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), Speed(4Bytes)

DataIn:

Index: 212 MOTOR_STOP_SPEED

Desc: Will change motor stopping speed. If reading back from get_long_data will not alter the value.

Units: pulse per second
Minimum: 100
Maximum: 20000 -- will attempt anything but would stay under
Default: 5000

If the top speed is less than the start speed, the start and stop speed will be set to this top speed setting and it will run at a constant speed.

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Speed(4Bytes)

Index: 212 MOTOR_STOP_SPEED

Desc: Will return the motor stopping speed.

Units: pulse per second
Minimum: 100
Maximum: 20000 -- will attempt anything but would stay under
Default: 5000

If the top speed is less than the start speed, the start and stop speed will be set to this top speed setting and it will run at a constant speed.

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), s-curveEnable(4Bytes)

DataIn:

Index: 213 RAMP_S_CURVE

Desc: Enables or disables the ramp s- curve. The s curve only occurs on ramp up if the acceleration > 20 and the speed is greater than the boost speed (default is the motor resolution). Set 0 to disable. 1 to enable.
Default is enabled.

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Speed(4Bytes)

Index: 213 RAMP_S_CURVE

Desc: Data determines whether ramp s curve is Enabled or disabled the ramp s-curve. The s curve only occurs on ramp up if the acceleration > 20 and the speed is greater than the boost speed (default is the motor resolution). Set 0 to disable. 1 to enable.

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), ServoRetry(4Bytes)

DataIn:

Index: 215 SERVO_RETRY

Desc: Sets the number of times the system will try and servo the motor to the target position after a move has been completed and before the busy status is released.

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), ServoRetry(4Bytes)

Index: 215 SERVO_RETRY

Desc: Gets the number of times the system will try and servo the motor to the target position after a move has been completed and before the busy status is released.

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), StaticGain(4Bytes)

DataIn:

Index: 217 GAIN_STATIC

Desc: Sets the motor static gain. The static gain is the gain used when the motor is idle (Motor powered but not moving).

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), StaticGain(4Bytes)

Index: 217 GAIN_STATIC

Desc: Gets the motor static gain. The static gain is the gain used when the motor is idle (Motor powered but not moving).

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), DymanicGain(4Bytes)

Index: 218 GAIN_DYNAMIC

Desc: Gets the motor Dymanic gain. The dymanic gain is the gain used when the motor is moving.

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), DymanicGain(4Bytes)

DataIn:

Index: 218 GAIN_DYNAMIC

Desc: Sets the motor Dymanic gain. The dymanic gain is the gain used when the motor is moving.

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), BoostGain(4Bytes)

DataIn:

Index: 219 GAIN_BOOST

Desc: Sets the motor boost gain. The boost gain is the gain used in servo mode. See SERVO_RETRY.

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), BoostGain(4Bytes)

Index: 219 GAIN_BOOST

Desc: Gets the motor boost gain. The boost gain is the gain used in servo mode. See SERVO_RETRY.

---

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), DerivativeGain(4Bytes)

Index: 220 GAIN_DERIVATIVE

Desc: Gets the motor's derivative gain. Also called the zero term.

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), DerivativeGain(4Bytes)

DataIn:

Index: 220 GAIN_DERIVATIVE

Desc: Sets the motor's derivative gain. Also called the zero term.

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), IntegralGain(4Bytes)

DataIn:

Index: 221 GAIN_INTEGRAL

Desc: Sets the motor's integral gain. Also called the pole term.

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), IntegralGain(4Bytes)

Index: 221 GAIN_INTEGRAL

Desc: Gets the motor's integral gain. Also called the pole term.

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Temperture(4Bytes)

Index: 222 MOTOR_TEMP

Desc: Gets the motor driver temperture. In degrees Farhenheight * 100.

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), MaxTemp(4Bytes)

Index: 225 MOTOR_MAX_TEMP

Desc: Gets the maximum allowable motor driver temperture value.
When the motor driver temperture exceeds this value the motor driver will generate a halt/crash condition. A value of zero disables this function. Default value is zero.

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), MaxTemp(4Bytes)

DataIn:

Index: 225 MOTOR_MAX_TEMP

Desc: Sets the maximum allowable motor driver temperture value.
When the motor driver temperture exceeds this value the motor driver will generate a halt/crash condition. A value of zero disables this function. Default value is zero.

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), MaxTemp(4Bytes)

DataIn:

Index: 226 MOTOR_MAX_CURRENT

Desc: Sets the maximum allowable motor driver current value.
When the motor driver current exceeds this value the motor driver will generate a halt/crash condition. A value of zero disables this function. Default value is zero. Units mA.

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), MaxTemp(4Bytes)

Index: 226 MOTOR_MAX_CURRENT

Desc: Gets the maximum allowable motor driver current value.
When the motor driver current exceeds this value the motor driver will generate a halt/crash condition. A value of zero disables this function. Default value is zero. Units mA.

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), ProportionalGain(4Bytes)

DataIn:

Index: 227 GAIN_PROPORTIONAL_SCALE

Desc: Sets the motor's proportional gain scale. Also called the pole term.

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), ProportionalGainScale(4Bytes)

Index: 227 GAIN_PROPORTIONAL_SCALE

Desc: Sets the motor's proportional gain scale. Also called the pole term.

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), IntegralGainScale(4Bytes)

Index: 228 GAIN_INTEGRAL_SCALE

Desc: Gets the motor's integral gain scale. Also called the pole term.

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), IntegralGain(4Bytes)

DataIn:

Index: 228 GAIN_INTEGRAL_SCALE

Desc: Sets the motor's integral gain scale. Also called the pole term.

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), DerivativeGain(4Bytes)

DataIn:

Index: 229 GAIN_DERIVATIVE_SCALE

Desc: Sets the motor's derivative gain scale. Also called the pole term.

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), DerivativeGainScale(4Bytes)

Index: 229 GAIN_DERIVATIVE_SCALE

Desc: Gets the motor's derivative gain scale. Also called the pole term.

---

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), MotorPower (4Bytes)

Index: 234 MOTOR_POWER

Desc: Gets the motor power value 1=on and 0=off.

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), MotorPower (4Bytes)

DataIn:

Index: 234 MOTOR_POWER

Desc: Sets the motor power.
Range: 0=Off, 1=Brake, 2-100% of full power.

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), EncoderMode (4Bytes)

DataIn:

Index: 235 ENCODER_MODE

Desc: Set the encode mode.
0=off, >0=On.

If sent to the single stepper motor module --
encoder 1 is default.
If the data = 2 -- will use alternative encoder.
4 = use newer absolute encoder

If encoder is set to off, soft limits are disabled as well.

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), EncoderMode (4Bytes)

Index: 235 ENCODER_MODE

Desc: Gets the encode mode,
0 = none
1 = interior (std encoder)
2 = external (come in second port of 6056)
4 = absolute encoder

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Status5000(4Bytes)

Index: 236 STATUS_5000

Desc: Gets the motor driver status in a configuration that mimics the MAC5000 status commands.

See MAC 6000 Reports

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), EndLimitLevel (4Bytes)

Index: 237 ENDLIMIT_LEVEL

Desc: Get the end limit active level. 0=Active Low, 1=Active High.

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), EndLimitLevel (4Bytes)

DataIn:

Index: 237 ENDLIMIT_LEVEL

Desc: Set the end limit active level. 0=Active Low, 1=Active High.

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes)

DataIn: Index(2Bytes), EndLimitLevel (4Bytes)

Index: 238 ENDLIMIT_ENABLE

Desc: Get the end function whether enabled or not. 0=Disabled, 1=Enabled. Currently can't control just one, it is all or nothing.

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes)

DataIn: Index(2Bytes), EndLimitLevel (4Bytes)

Index: 238 ENDLIMIT_ENABLE

Desc: Set the endlimit function whether enabled or not. 0=Disabled, 1=Enabled. Currently can't control just one, it is all or nothing.

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), TrackBallMode (4Bytes)

Index: 239 TRACKBALL

Desc: Get the trackball mode. 0=off, 1=on.

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), TrackBallMode (4Bytes)

DataIn:

Index: 239 TRACKBALL

Desc: Set the trackball mode. 0=off, 1=on.

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), SoftLimitMode(4Bytes)

DataIn:

Index: 240 SOFT_LIMIT_MODE

Desc: Enables/disables soft limit mode. Bit 0 controls the LOW soft limit and Bit 1

controls the HIGH soft limit. 0=off, 1=0n. Examples: 0= Both off, 1=low limit on, 2=high limit on and 3=both soft limits on.

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), SoftLimitMode(4Bytes)

Index: 240 SOFT_LIMIT_MODE

Desc: Gets soft limit mode. Bit 0 controls the LOW soft limit and Bit 1 controls the HIGH soft limit. 0=off, 1=0n. Examples: 0= Both off, 1=low limit on, 2=high limit on and 3=both soft limits on.

---

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), ServoCheckMode(4Bytes)

Index: 241 SERVO_CHECKING

Desc: Gets the servo checking mode. 0=off, 1=on.

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), ServoCheckMode(4Bytes)

DataIn:

Index: 241 SERVO_CHECKING

Desc: Sets the servo checking mode. 0=off, 1=on. Enabling this feature will cuase the system to hold/track position in motor idle mode.

---

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), HomeCapture(4Bytes)

Index: 242 HOME_CAPTURE

Desc: Gets the home capture mode. o=off, 1=on.

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), HomeCapture(4Bytes)

DataIn:

Index: 242 HOME_CAPTURE

Desc: Sets the home capture mode. o=off, 1=on.
Currently does not save in flash will always boot up off til set again.

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), HomePulse(4Bytes)

DataIn:

Index: 243 HOME_PULSE

Desc: Sets the home signal active mode. o=low 1=high.
Currently this can not be saved in flash, default is low.

84 GET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), HomePulseMode(4Bytes)

Index: 243 HOME_PULSE

Desc: Gets the home signal active mode. o=low 1=high.

---

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), JoyStickMode(4Bytes)

Index: 245 JOYSTICK

Desc: Gets the joystick mode. o=off, 1=on.

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), JoyStickMode(4Bytes)

DataIn:

Index: 245 JOYSTICK
Desc: Sets the joystick mode. o=off, 1=on.

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)
DataIn: Index(2Bytes), JoyStickRevMode(4Bytes)
Index: 246 JOYSTICK_DIRECTION_REV
Desc: Gets the joystick reverse mode. o=normal, 1=reversed

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), JoyStickRevMode(4Bytes)
DataIn:
Index: 246 JOYSTICK_DIRECTION_REV
Desc: Sets the joystick reverse mode. o=normal, 1=reversed

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)
DataIn: Index(2Bytes), MotorRevMode(4Bytes)
Index: 248 MOTOR_DIRECTION_REV
Desc: Get the motor direction revserse mode. 0=normal, 1=reversed.

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), MotorRevMode(4Bytes)
DataIn:
Index: 248 MOTOR_DIRECTION_REV
Desc: Set the motor direction reverse mode. 0=normal, 1=reversed.

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), JoyStickMode(4Bytes)

Index: 249 CW_ENDLIMIT_POS

Desc: Get the position of the last cw endlimit detected.

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), JoyStickMode(4Bytes)

Index: 250 CCW_ENDLIMIT_POS

Desc: Get the position of the last ccw endlimit detected.

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), BlinkingMode (4Bytes)

DataIn:

Index: 251 BLINK_LEDS

Desc: When mode = on, blinks the leds for that module. 0=off, 1=on. The leds will be shut off before any move command is acted on. For the dc and stepper motors the endlimits will blink so the user can distinguish one module (motor) from the other.

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), Shutter (2 bytes), n/a(2Bytes)

DataIn: Index(2Bytes), Shutter (2 bytes),Exposure Time(2 Bytes)

Index: 252 SHUTTER_EXPOSURE

Desc: Will change shutter exposure time. The most significant 2 bytes of the data will give the shutter, the least significant 2 bytes will give the shutter exposure value in milliseconds.

If reading back from get_long_data will not alter the value.

The most significatn 2 bytes indicates the shutter

FilterA will interpret
0=>shutter1
1=>shutter3
100=>shutter1
101=>shutter2
102=>shutter3

FilterB will interpret
0=>shutter2
1=>shutter3
100=>shutter1
101=>shutter2
102=>shutter3

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), Shutter (2 bytes) Exposure Time(2 Bytes)

DataIn:

Index: 252 SHUTTER_EXPOSURE

Desc: Will change shutter exposure time. The most significant 2 bytes of the data will give the shutter, the least significant 2 bytes will give the shutter exposure value in milliseconds.

If reading back from get_long_data will not alter the value.

The most significatn 2 bytes indicates the shutter

FilterA will interpret
0=>shutter1
1=>shutter3
100=>shutter1
101=>shutter2
102=>shutter3

FilterB will interpret
0=>shutter2
1=>shutter3
100=>shutter1
101=>shutter2
102=>shutter3

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), Shutter (2 bytes), Control Bits(2 Bytes)

DataIn:

Index: 253 SHUTTER_CONTROL

Desc: The Most Significant 2 bytes of data select the shutter.
The Least significant 2 bytes Will change shutter control information.

The most significant 2 bytes indicates the shutter index
Main MotorA will interpret
0=>shutter1
1=>shutter3
100=>shutter1
101=>shutter2
102=>shutter3

MotorB will interpret
0=>shutter2
1=>shutter3
100=>shutter1
101=>shutter2
102=>shutter3

bit 0: SHUTTER_MODE 0 -- manual shutter mode -- shutters don't move while moving filter wheel unless cmd to open comes
1 -- auto shutter mode - shutter will close during moves and open at the end of a move

bit 1: EXPOSURE_MODE 0 - no auto shutter movement
1 - shutters expose at end of move

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Shutter (2Bytes), ControlBits (2Bytes)

Index: 253 SHUTTER_CONTROL

Desc: The Most Significant 2 bytes of data select the shutter.
The Least significant 2 bytes Will change shutter control information.

The most significant 2 bytes indicates the shutter index
Main MotorA will interpret
0=>shutter1
1=>shutter3
100=>shutter1
101=>shutter2

102=>shutter3

MotorB will interpret
0=>shutter2
1=>shutter3
100=>shutter1
101=>shutter2
102=>shutter3

bit 0: SHUTTER_MODE 0 -- manual shutter mode -- shutters don't move while moving filter wheel unless cmd to open comes
1 -- auto shutter mode - shutter will close during moves and open at the end of a move

bit 1: EXPOSURE_MODE 0 - no auto shutter movement
1 - shutters expose at end of move

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), VoltageRating(4Bytes)

DataIn:

Index: 254 MOTOR_VOLTAGE_RATING

Desc: Will change motor voltage rating. This value effects the saturation duty cylce of the pwm. It will also possibly be saved in flash. The default value is 19 volts. The max value is 24. If reading back from get_long_data will not alter the value.

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), VoltageRating(4Bytes)

Index: 254 MOTOR_VOLTAGE_RATING

Desc: Will read motor voltage rating. This value effects the saturation duty cylce of the pwm. It will also possibly be saved in flash. The default value is 19 volts. The max value is 24. If reading back from get_long_data will not alter the value.

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), FIRMWARE_MODE (4Bytes)

DataIn:

Index: 255 FIRMWARE_MODE

Desc: Causes the firmware to jump to either the module boot code or application code. Calling this command will cause the module to reset.
0 = Applications firmware (default)
1 = Boot/Diagnostic firmware

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), FIRMWARE_MODE (4Bytes)

Index: 255 FIRMWARE_MODE

Desc: Returns the current mode of the firmware
0 = Applications firmware (default)
1 = Boot/Diagnostic firmware

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), Shutter(2Bytes),OPEN_COAST(1 Bytes),OPEN_KICK(1Byte)

DataIn:

Index: 256 SHUTTER_OPEN_TIME

Desc: Most Significant 2 bytes of data select the shutter.
The least significant 2 bytes will change shutter open kick and coast time.

The high 2 bytes indicates the shutter index
Main MotorA will interpret
0=>shutter1
1=>shutter3
100=>shutter1
101=>shutter2
102=>shutter3

MotorB will interpret
0=>shutter2
1=>shutter3
100=>shutter1
101=>shutter2
102=>shutter3

The third byte will give the open coast byte and the lsb will give the open kick value (all in milliseconds).
If reading back from get_long_data will not alter the value.

SENDING
CAN 17 83 256 6553512 (0x64000C) sets the shutter open time to 12 milliseconds

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), OPEN_COAST(1Byte), OPEN_KICK(1Byte)

Index: 256 SHUTTER_OPEN_TIME

Desc: Most Significant 2 bytes of data select the shutter.
The least significant 2 bytes will change shutter open kick and coast time.

The high 2 bytes indicates the shutter index
Main MotorA will interpret
0=>shutter1
1=>shutter3
100=>shutter1
101=>shutter2
102=>shutter3

MotorB will interpret
0=>shutter2
1=>shutter3
100=>shutter1
101=>shutter2
102=>shutter3

The third byte will give the open coast byte and the lsb will give the open kick value (all in milliseconds).
If reading back from get_long_data will not alter the value.
SENDING
CAN 17 83 256 6553512 (0x64000C) sets the shutter open time to 12 milliseconds

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), CLOSE_COAST(1Byte), CLOSE_KICK(1Byte)

Index: 257 SHUTTER_CLOSE_TIME

Desc: Least Significant 2 bytes of data select the shutter.
The Most significant 2 bytes will change shutter close kick and close coast time.

The low 2 bytes indicates the shutter index
Main MotorA will interpret
0=>shutter1
1=>shutter3
100=>shutter1
101=>shutter2
102=>shutter3

MotorB will interpret
0=>shutter2
1=>shutter3
100=>shutter1
101=>shutter2
102=>shutter3

The third byte will give the close coast byte and the least significant byte will give the close kick value (all in milliseconds).
If reading back from get_long_data will not alter the value.

---

83 SET_LONG_DATA - ***Set long data command***

DataOut : Index(2Bytes), Shutter(2Bytes), CLOSE_COAST(2Bytes), CLOSE_KICK(2Bytes)

DataIn:

Index: 257 SHUTTER_CLOSE_TIME

Desc: Least Significant 2 bytes of data select the shutter.
The Most significant 2 bytes will change shutter close kick and close coast time.

The low 2 bytes indicates the shutter index
Main MotorA will interpret
0=>shutter1
1=>shutter3
100=>shutter1
101=>shutter2
102=>shutter3

MotorB will interpret
0=>shutter2
1=>shutter3
100=>shutter1
101=>shutter2

102=>shutter3

The third byte will give the close coast byte and the least significant byte will give the close kick value (all in milliseconds).
If reading back from get_long_data will not alter the value.

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), Shutter (2 bytes) Shutter position(2bytes)

DataIn:

Index: 258 SHUTTER_POSITION

Desc: Moves the shutter open, closed or exposed indicated by the data.
The most significant two bytes of the data is the index which indicates the shutter number. The index is zero based.

FilterA will interpret:
0=>shutter1
1=>shutter3
100=>shutter1
101=>shutter2
102=>shutter3

FilterB will interpret
0=>shutter2
1=>shutter3
100=>shutter1
101=>shutter2
102=>shutter3

The least significant two bytes is the shutter move command
If the command == 0 Close Shutter
command == 1 Open Shutter
command == 2 Expose Shutter (open for pre-determined time then close)

84 GET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), Shutter (2 bytes) n/a (4Bytes)

DataIn: Index(2Bytes), Shutter (2 bytes) Shutter Position(0-close,1-open,2-exp)

Index: 258 SHUTTER_POSITION

Desc: Returns the shutter current status.
0 = Shutter Closed
1 = Shutter Opened

2 = Shutter Exposing (open for pre-determined time then close)

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes0), Not Used(2Bytes), backlash value(2Bytes)

DataIn:

Index: 259 BACKLASH_COMP

Desc: Default is 0. If a backlash compensation value is set, the motor will approach all requested targets (not joystick or trackball or servo) from a direction and distance of this setting. If no compensation is necessary because motor is approaching from correct direction nothing will occur. If coming from opposite direction, motor will progress past the target the number of motor steps set here and come back to target (if closed loop, number of encoder steps).

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Not Used(2Bytes), Back Lash compensation value (2Bytes)

Index: 259 BACKLASH_COMP

Desc: FOR FUTURE - Not Implemented Yet

Default is 0. If a backlash compensation value is set the motor will approach all requested targets (not joystick or trackball or servo) from a direction and distance of this setting. If no compensation is necessary because motor is approaching from correct direction nothing will occur. If coming from opposite direction, motor will progress past the target the number of motor steps set here and come back to target (if closed loop, number of encoder steps).

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), Position(4Bytes)

DataIn:

Index: 260 MOTOR_SECOND_ENCODER

Desc: Sets the second motor encoder. If its a dual stepper motor it will set the other motor's encoder.

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Position(4Bytes)

Index: 260 MOTOR_SECOND_ENCODER

Desc: Gets the second motor encoder. Be aware if this is a dual motor board it will get the other motor's encoder!

---

84 GET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Configuration (4Bytes)

Index: 262 LIMIT_OUT_LEVEL

Desc: Gets the current value of the limit out signal (J3-6) on the synch connector

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), Configuration (4Bytes)

DataIn:

Index: 262 LIMIT_OUT_LEVEL

Desc: Manually sets the output level of the limit out signal.
If data > 0 signal is high else low.

---

84 GET_LONG_DATA - *Get long data command.*

DataOut: Index(2Bytes), CommandNumber(2Bytes), IndexNumber(2Bytes)

DataIn: Index(2Bytes), CommandVersion (4Bytes)

Index: 263 IS_CMD_AVAIL

Desc: Returns zero if the module does not support the command/index passed. Otherwise returns non-zero to indicate that the command is supported. The non-zero value returned should start at a 1 and increment if command changes so that the user can tell the version level of the command.

---

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Configuration (4Bytes)

Index: 264 SYNCH_OUT_WIDTH

Desc: If the synch out is configured to be pulsed not level this will alter the pulse width. The synch out signal will be pulsed and the pulse width will be the data here times 50 microseconds.
The default is 13, so the pusle width will be 650 microseconds.

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), Configuration (4Bytes)

DataIn:

Index: 264 SYNCH_OUT_WIDTH

Desc: If the synch out is configured to be pulsed not level this will alter the pulse width. The synch out signal will be pulsed and the pulse width will be the data here times 50 microseconds.
The default is 13, so the pulse width will be 650 microseconds. Value is 0 - 0xff

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), EncoderMode (4Bytes)

DataIn:

Index: 265 MAINTAIN_RATIO

Desc: If set the current ratio setting in memory is maintained and no encoder check is done.

0 - Default - do encoder check
1 - No encoder check

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), EncoderMode (4Bytes)

Index: 265 MAINTAIN_RATIO

Desc: Get Ratio Check Mode
If set the current ratio setting in memory is maintained and no encoder check is done.

0 - Default - do encoder check
1 - No encoder check

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), FilterWheelSlowMode (4Bytes)

DataIn:

Index: 266 FW_SLOW_MODE

Desc: Set the fw to slow speed, 0=off, 1=On.

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), FilterWheelSlowMode (4Bytes)

Index: 266 FW_SLOW_MODE

Desc: Get the fw to slow speed, 0=off, 1=On.

---

83 SET_LONG_DATA - ***Set Long Data Command***

DataOut: Index(2Bytes), Piezo Control Source(4Bytes)

DataIn:

Index: 267 PIEZO_CONTROL_SOURCE

Desc: If set to 0 - Will use Interface to control position and report position.

If set to 1 - Will use A/D channel 1 as the desired target
- Will use D/A channel 1 as the current position

If set to 1 and running open loop
- Will use A/D channel 1 as the desired target.
Value will be placed on D/A directly.
- D/A channel 1 will reflect the current strain- guage position

---

83 SET_LONG_DATA - ***Set Long Data Command***

DataOut: Index(2Bytes), Piezo Control Source(4Bytes)

DataIn:

Index: 267 PIEZO_CONTROL_SOURCE

Desc: Lower 3 bits control source
If set to 0 - Will use Interface to control position and report position.

If set to 1 - Will use A/D channel 1 as the desired target
- Will use D/A channel 1 as the current position

If set to 1 and running open loop
- Will use A/D channel 1 as the desired target.
Value will be placed on D/A directly.
- D/A channel 1 will reflect the current strain- guage position

If set to 2 will base movement on A/D Channel 1 as a +/- 2.5V error signal.
Only used if module supports FOCUSTRACK

Bit 7 must also be set to allow operation in FOCUSTRACK Mode.

NOTE: FOCUSTRACK operation is a special case and module needs modification to support it.

84 GET_LONG_DATA - *Get Long Data Command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Piezo Control Source(4Bytes)

Index: 267 PIEZO_CONTROL_SOURCE

Desc: Lower 3 bits control source
If set to 0 - Will use Interface to control position and report position.

If set to 1 - Will use A/D channel 1 as the desired target
- Will use D/A channel 1 as the current position

If set to 1 and running open loop
- Will use A/D channel 1 as the desired target.
Value will be placed on D/A directly.
- D/A channel 1 will reflect the current strain- guage position

If set to 2 will base movement on A/D Channel 1 as a +/- 2.5V error signal.
Only used if module supports FOCUSTRACK

Bit 7 must also be set to allow operation in FOCUSTRACK Mode.

NOTE: FOCUSTRACK operation is a special case and module needs modification to support it.

83 SET_LONG_DATA - ***Set Long Data Command***

DataOut: Index(2Bytes), Piezo Control Source(4Bytes)

DataIn:

Index: 267 PIEZO_CONTROL_SOURCE

Desc: If set to 0 - Will use Interface to control position and report position.

If set to 1 - Will use A/D channel 1 as the desired target
- Will use D/A channel 1 as the current position

If set to 1 and running open loop
- Will use A/D channel 1 as the desired target.
Value will be placed on D/A directly.
- D/A channel 1 will reflect the current strain- guage position

---

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), A/D channel(1Byte), n/a(3Bytes)

DataIn: Index(2Bytes), A/D channel(1Byte), A/D reading(3Bytes)

Index: 268 ANALOG_INPUT_BINARY

Desc: 1st byte specifies channel.
0-Channel 0
1-Channel 1

Return value is given in 24bit format

---

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), DA channel(1Byte), n/a(3Bytes)

DataIn: Index(2Bytes), DA channel(1Byte), DA value(3Bytes)

Index: 269 ANALOG_OUTPUT_BINARY

Desc: Most Significant Byte
Bits 0-3 specifies D/A Channel
Valid Range 0-7 for DAIO board
Valid Range 0-1 for PIEZO board

Bit 4 - If set then the return value in Least Significant 3 bytes specify power on default. (Only Available on DAIO board)

Bytes 2-4: Don't Care

Returns either Current D/A value or power on value in 24 bit format.

---

83 SET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), DA channel(1Byte), DA value(3Bytes)

DataIn:

Index: 269 ANALOG_OUTPUT_BINARY

Desc: Most Significant Byte
Bits 0-3 specifies D/A Channel
Valid Range 0-7 for DAIO board
Valid Range 0-1 for PIEZO board

Bit 4 - If set then the return value in Least Significant 3 bytes specify power on default. (Only Available on DAIO board)

Bytes 2-4: Don't Care

Returns either Current D/A value or power on value in 24 bit format.

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), AD channel(1Byte), Offset value(3Bytes)

DataIn:

Index: 270 CALIB_A2D_OFFSET

Desc: Most Significant byte specifies channel.
0-Channel 0
1-Channel 1

Offset to add to A/D value before processing.
value can range from +65535 to -65536

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), AD channel(1Byte), n/a(3Bytes)

DataIn: Index(2Bytes),AD chanel(1Byte), Offset value(3Bytes)

Index: 270 CALIB_A2D_OFFSET

Desc: Most Significant byte specifies channel.
0-Channel 0
1-Channel 1

Offset to add to A/D value before processing.
value can range from +65535 to -65536

---

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), AD channel(1Byte), n/a(3Bytes)

DataIn: Index(2Bytes), AD channel(1Byte), Gain value(3Bytes)

Index: 271 CALIB_A2D_GAIN

Desc: 1st byte specifies channel.
0-Channel 0
1-Channel 1

Gain Value is specified as gain*10000.
So for a gain of 1 the value would be 10000
For a gain of 1.01 the value would be 10100

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), AD channel(1Byte), Gain value(3Bytes)

DataIn:

Index: 271 CALIB_A2D_GAIN

Desc: Most Significant byte specifies channel.
0-Channel 0
1-Channel 1

Gain Value is specified as gain*10000.
So for a gain of 1 the value would be 10000
For a gain of 1.01 the value would be 10100

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), SGS_MAX(4Bytes)

DataIn:

Index: 272 PIEZO_SGS_MAX

Desc: This value is stored in the one-wire for each piezo stage.
This is the calibration value for the stage.

It translates to the number of A/D counts for a full scale movement of the stage. Position is computed by:
POS=A2D*65536/SGS_MAX+POS_OFFSET.

Typical value for SGS_MAX is 22000 - 25000

POS_OFFSET is set when the HERE command is issued.

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), SGS_MAX(4Bytes)

Index: 272 PIEZO_SGS_MAX

Desc: This value is stored in the one-wire for each piezo stage.
This is the calibration value for the stage.

It translates to the number of A/D counts for a full scale movement of the stage. Position is computed by:
POS=A2D*65536/SGS_MAX+POS_OFFSET.

Typical value for SGS_MAX is 22000 - 25000

POS_OFFSET is set when the HERE command is issued.

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), Port(2Bytes), value(2Bytes)

DataIn:

Index: 273 GPIO_CONTROL

Desc: Most Significant two bytes (PORT) specifies what to write to.
0 - Logic I/O Input Register
1 - Logic I/O Input Latch Register
2 - Logic I/O Output Register
3 - Logic I/O Direction Register
4 - Switch I/O Output Register
5 - Switch I/O Status Register
0x102 - Power Up Output for Logic I/O Output Register
0x103 - Power Up value for Logic I/O Direction Register
0x014 - Power Up value for Switch I/O Register
0x200 - Resets all registers to power up defaults

Least Significant two bytes (VALUE) is the value written to the port specified.

See DAIO controller documentation for description of ports

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), Port(2Bytes), n/a(2Bytes)

DataIn: Index(2Bytes),Port(2Bytes), value(2Bytes)

Index: 273 GPIO_CONTROL

Desc: Most Significant two bytes (PORT) specifies what to read from.
0 - Logic I/O Input Register
1 - Logic I/O Input Latch Register
2 - Logic I/O Output Register
3 - Logic I/O Direction Register
4 - Switch I/O Output Register
5 - Switch I/O Status Register
0x102 - Power Up Output for Logic I/O Output Register
0x103 - Power Up value for Logic I/O Direction Register
0x014 - Power Up value for Switch I/O Register
0x200 - Resets all registers to power up defaults

Least Significant two bytes (VALUE) is a don't care on a read operation.

See DAIO controller documentation for description of ports.

---

83 SET_LONG_DATA - *Set long data command*

DataOut: Index(2Bytes), Value(4Bytes)

DataIn:

Index: 274 PWM_PERIOD

Desc: Value is the Period in 0.1uS Steps
A period of 3ms would be a value of 30000

Valid Range is 85 to 87000 or 8.5uS to 87ms

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Value(4Bytes)

Index: 274 PWM_PERIOD

Desc: Value is the Period in 0.1uS Steps
A period of 3ms would be a value of 30000

Valid Range is 85 to 87000 or 8.5uS to 87ms

---

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Value(4Bytes)

Index: 275 PWM_CONTROL

Desc: Contains the control bits for the PWM outputs of the EDAIO board.

Bit 0 - Start/Stop PWM 1
Bit 1 - Level PWM 1. 0 indicates Normally Low
Bit 4 - Start/Stop PWM 2
Bit 5 - Level PWM 2. 0 indicates Normally Low
Bit 7 - Enable

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), Value(4Bytes)

DataIn:

Index: 275 PWM_CONTROL

Desc: Contains the control bits for the PWM outputs of the EDAIO board.

Bit 0 - Start/Stop PWM 1
Bit 1 - Level PWM 1. 0 indicates Normally Low
Bit 4 - Start/Stop PWM 2
Bit 5 - Level PWM 2. 0 indicates Normally Low
Bit 7 - Enable

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), Channel(1 Byte), value(3Bytes)

DataIn:

Index: 276 PWM_DUTY

Desc: channel is either 0 or 1
value is the Duty cycle in 0.01% increments
50% would be 5000

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), channel(1 Byte), n/a(3Bytes)

DataIn: Index(2Bytes),channel(1 Byte), value(3Bytes)

Index: 276 PWM_DUTY

Desc: chanel is either 0 or 1
value is the Duty cycle in 0.01% increments
50% would be 5000

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), Joystick Deflection (4Bytes)

DataIn:

Index: 277 JOYSTICK_EXP_FACTOR

Desc: This is the exponential factor used in the joystick deflection equation when the joystick is moved without use of the slew button. The default value is 60. The higher the value the longer the slow movements under deflection. As the exp factor approaches 1 the movement of the joystick becomes more linear.

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), n/a(4Bytes)

DataIn: Index(2Bytes), Joystick Deflection (4Bytes)

Index: 277 JOYSTICK_EXP_FACTOR

Desc: This is the exponential factor used in the joystick deflection equation when the joystick is moved without use of the slew button. The default value is 60. The higher the value the longer the slow movements under deflection. As the exp factor approaches 1 the movement of the joystick becomes more linear.

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), Shutter (2 bytes) Voltage Level(2Bytes)

DataIn:

Index: 278 SHUTTER_VOLT_LEVEL

Desc: Most Significant 2 bytes of data select the shutter.
The least significant 2 bytes will set the voltage level.
0-Low
1-High

The most significant 2 bytes indicate the shutter index
Main MotorA will interpret
0=>shutter1
1=>shutter3
100=>shutter1
101=>shutter2
102=>shutter3

MotorB will interpret
0=>shutter2
1=>shutter3
100=>shutter1
101=>shutter2
102=>shutter3

SENDING
CAN 17 83 278 6553601 (0x640001) sets the voltage level to high.

84 GET_LONG_DATA - *Get long data command*

DataOut: Index(2 Bytes), Shutter (2 bytes), n/a(2Bytes)

DataIn: Index(2 Bytes), Shutter (2 bytes), Voltage Level(2Bytes)

Index: 278 SHUTTER_VOLT_LEVEL

Desc: Most Significant 2 bytes of data select the shutter.
The least significant 2 bytes will set the voltage level.
0-Low
1-High

The high 2 bytes indicates the shutter index
Main MotorA will interpret
0=>shutter1
1=>shutter3
100=>shutter1
101=>shutter2
102=>shutter3

MotorB will interpret
0=>shutter2
1=>shutter3
100=>shutter1
101=>shutter2
102=>shutter3

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), RecordID (4Bytes)

DataIn:

Index: 279 RECORD_ID

Desc: Will change the RecordID used for default PID calculation in DC Motor Driver Modules

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), n/a (4Bytes)

DataIn: Index(2Bytes), RecordID (4Bytes)

Index: 279 RECORD_ID

Desc: Will read the RecordID used for default PID calculation in DC Motor Driver Modules

---

83 SET_LONG_DATA - ***Set long data command***

DataOut: Index(2Bytes), Data (4Bytes)

DataIn:

Index: 280 JOYSTICK_3D_PARAMATER

Desc: Will change the 3d joystick paramaters

High 2 bytes of data will determine paramater numbers
Low 2 bytes of data will determine paramater value

Par # Description
0 xy factor 1-100 1 (1-lin 100-high expo)
1 z factor 1-100 1
2 slew xy factor 1-100 30
3 slew z factor 1-100 1
4 xy deadband 1-30 10 (in %)
5 slew xy deadband 1-30 30 (in %)
6 z deadband 1-30 10 (in %)
7 mode 0 or 1 0 0-rot=slew 1-tr=slew)
8 slew point 10-60 50 (in %)

84 GET_LONG_DATA - ***Get long data command***

DataOut: Index(2Bytes), Data (4Bytes)

DataIn: Index(2Bytes), Data (4Bytes)

Index: 280 JOYSTICK_3D_PARAMATER

Desc: Will Get the 3d joystick paramaters

High 2 bytes of data will determine paramater numbers
Low 2 bytes of data will determine paramater value

Par # Description
0 xy factor 1-100 1 (1-lin 100-high expo)
1 z factor 1-100 1
2 slew xy factor 1-100 30
3 slew z factor 1-100 1
4 xy deadband 1-30 10 (in %)
5 slew xy deadband 1-30 30 (in %)
6 z deadband 1-30 10 (in %)
7 mode 0 or 1 0 0-rot=slew 1-tr=slew)
8 slew point 10-60 50 (in %)

MAC6000 CAN Enumeration Protocol

07/21/2006 Version 0.5 Preliminary

Description

This document describes the MAC6000 CAN enumeration protocol.

Definitions

Node Id - CAN Address Nodes (Node Id are not related to Device Numbers)
Device Number – A number representing the LEP device letter. 1=X, 2=Y, 3=Z …32
Device Id – A letter representing the LEP device module eg. X, Y, Z …

CAN ID Nodes:

0 = Global Commands
1 = Master Interface
2-31 = All other modules.

Boot Up State

On boot up, all modules must boot up with the global node reception enabled. Modules must be able to receive CAN commands within 1 second after reset.
The global node id is zero. No other node id may be enabled.
There is one exception to this rule, the master interface will boot up with a node id of one.

After the module has initialized it should send a REPORT_BOOTUP message to the global node id. This will notify all modules on the CAN bus that the module is present on the bus.

Enumeration

On boot up the master interface will send a WHOS_OUT_THERE command. All modules present on the CAN bus must response to this command within 1 seconds with the REPORT_BOOTUP reoprt. The module will response with it's current node id and the unique serial number.

Upon receiving the WHOS_OUT_THERE response the master interface will assign unique node id to all the modules that responded to the WHOS_OUT_THERE command. The interface will use the SET_NODE_ID command along with the module's unique serial number to set the unique node id. When a module receives the SET_NODE_ID command with a node id of non-zero the module is considered configured. Module's whose serial numbers are zero will not be configured. In configured state each module will response to commands on both the global node (zero) and it's assigned node id. The node ids are assigned arbitrarily starting at 2.

Module Mapping
Next the interface will request the module's type information. This is done using command GET_LONG_DATA with index of DEVICE_TYPE.

The interface will wait for a period of 2 seconds to gather all the module's DEVICE_TYPE information. After the 2 seconds the interface will map all the present modules.

Mapping consists of assigning a unique device number to each module. Device numbers are 1 to 32, 1=X, 2=Y and so on. Device numbers are assigned using the SET_LONG_DATA/DEVICENUM command. The module need only save this device number and represent it when queried.

Device numbers are assigned with the following priority. (Subject to change)

1 – Previous enumerated devices will be set with the same device number. The interface will save a copy of the last enumeration data (serial number and device number) on each shut down. If a module was present at the last shut down it will be mapped with the same device number.

2 – Next modules will be mapped by there requested device number that was returned in the DEVICE_TYPE command. If the requested device number is available the module will be assigned to it.

3 – Next the interface will try and assign device numbers based on device type returned on the DEVICE_TYPE command. If the device number solts are available the module will be assigned to it.

4 – All other modules that have unknown device types or the device type slot is fill will be assigned device number of 21 and up.

Type Num	**DeviceType Desc**	**Device Numbers**	**Device Letters**
1,2,3,4,5,6,7,8	Motor Driver (DC or Stepper)	1,2,3,4,5,6,7	X,Y,B,R,C,Z,T
11,17,18	I/O Controllers	8,9	I,O
15,16	Flat Finder	10	F
12,13,14	Focus Controller	11	F
19	Photometer Controller	12	P
9,10	Filter/Shutter Controller	17,18,19,20	S,S1,S2,S3

Modules who response to WHOS_OUT_THERE command after the 2 second period will still be mapped but there desired mapped position may not be granted.

After the interface assigns the device number the interface will assign the device number and request the VERSION_NO and MODULE_DATE.

Bus Response

All modules must response to a bus command with in X(TBA) seconds. If a module fails to response to a command with in this time the interface will mark this module as offline and all commands to the module will be rejected. If a module knows it will not able to response to a bus command it should send to the REPORT_ONLINE command.

Sample Enumeration:

From(node)	To(node)	Command (*response*)	Data	Description
Interface(1)	Global(0)	WHOS_OUT_THERE		Interface requests all modules to report there presents on the can bus.
6050(0)	Interface(1)	*WHOS_OUT_THERE*	0,0,1000	Stepper motor driver response with serial number (1000) is present with a currently node id of zero.
6051(0)	Interface(1)	*WHOS_OUT_THERE*	0,0,1001	DC motor driver response with serial number (1001) is present with a currently node id of zero.
Interface(1)	Global(0)	SET_NODE_ID	0,2,1000	Interface tells module with a serial number=1000 to change it's node id to 2.
Interface(1)	6050(2)	GET_LONG_DATA	Index=DEVICE_TYPE	Interface requests the device from the module.
Interface(1)	Global(0)	SET_NODE_ID	0,2,1000	Interface tells module with a serial number=1000 to change it's node id to 2.
Interface(1)	6051(3)	GET_LONG_DATA	Index=DEVICE_TYPE	Interface requests the device from the module.
6050(2)	Interface(1)	*GET_LONG_DATA*	Index=DEVICE_TYPE Data = device type	6050(2) response with its device type
6051(3)	Interface(1)	*GET_LONG_DATA*	Index=DEVICE_TYPE Data = device type	6051(3) response with its device type
Interfaces waits for 2 seconds, then maps the present modules – 6050(2) only show for brevity				

Interface(1)	6050(2)	SET_LONG_DATA	Index=DEVICENUM Data= Device Number	Interface asks 6050(2) for its name string.
Interface(1)	6050(2)	GET_LONG_DATA	Index=VERSION_NO	Interface asks 6050(2) for its version
6050(1)	Interface(1)	*GET_LONG_DATA*	Index=VERSION_NO Data = Module Ver	6050(2) response with its version
Interface(1)	6050(2)	GET_LONG_DATA	Index=VERSION_NO	Interface asks 6050(2) for its date
6050(2)	Interface(1)	*GET_LONG_DATA*	Index=VERSION_NO Data = Module Date	6050(2) response with its date.
Example of position request				
Interface(1)	6050(2)	GET_LONG_DATA	Index=POSITION	Interface requests modules position
6050(2)	Interface(1)	*GET_LONG_DATA*	Index=POSTION DATA= Position	6050(2) responses with it's position

```
enum DEVICE_TYPE  {
   DEVICE_TYPE_UNDEF,       // 0  - Undefined
   DEVICE_TYPE_EMOT ,       // 1  - Stepper Motor - 1.8 degree
   DEVICE_TYPE_EMOT_400,    // 2  - Stepper Motor - 0.9 degree
   DEVICE_TYPE_EMOTM,       // 3  - Stepper Motor - Mapper
   DEVICE_TYPE_SLIDE_STEP,  // 4  - Stepper Motor - Carousal
   DEVICE_TYPE_EMOTD,       // 5  - DC Motor -
   DEVICE_TYPE_EMOTS,       // 6  - DC Motor - Mapper
   DEVICE_TYPE_EMOTB,       // 7  - DC Motor - Brake
   DEVICE_TYPE_SLIDE_DC,    // 8  - DC Motor - Carousal
   DEVICE_TYPE_EFILS_STEP,  // 9  - Filter wheel - Stepper
   DEVICE_TYPE_EFILS_DC,    // 10 - Filter wheel - DC
   DEVICE_TYPE_PIEZO,       // 11 - Piezo driver
   DEVICE_TYPE_EAFC,        // 12 - Auto Focus Contoller
   DEVICE_TYPE_EAFC_STEP,   // 13 - Auto Focus - Stepper
   DEVICE_TYPE_EAFC_DC,     // 14 - Auto Focus - DC
   DEVICE_TYPE_FFIND_STEP,  // 15 - Flat Finder - Stepper
   DEVICE_TYPE_FFIND_DC,    // 16 - Flat Finder - DC
   DEVICE_TYPE_EDAIO,       // 17 - Digital Analog input output
   DEVICE_TYPE_NOSE ,       // 18 - Turret Changer
   DEVICE_TYPE_HPHCD,       // 19 - Photometer
   DEVICE_TYPE_INT          // 20 - Interface
   };
```

MAC 6000 Reports

Version 1.01 May 9 2007 **Status: Preliminary**

Report Guidelines:

A report contains events and status information that can change asynchronous. Reports can transmitted via three sources. They can be requested by another module using the REQUEST_REPORT(an index into a SET_LONG_DATA/GET_LONG_DATA) command. They can also be setup to be sent at timed increments or they can be sent unsolicted due to a change in pertinent information that they convey. The events or status information should be limited to information that the user would find useful. An example would be an error report or move finishing and the position change report.

Report Purpose:

The purpose of the report based system is to reduce polling, therefore reducing the communication traffic. This event based system is more efficient. With a report based architecture all pertinent information is sent asynchronously from the modules to the interface. Information that is not important to the end client will be shut off. When enabled this information is in turn sent up to a DLL or other moderator program that will further distribute the report to the required software modules. There are currently nine reports. These reports can be sent asynchronously or can be requested to be sent by any module. This allows other modules to sync to module. See the commands REQUEST_REPORT.

Report Types:

REPORT_BOOTUP - is sent when a module boots up, useful if module can be hot booted, or even interface and module get out of synch
REPORT_ONLINE - Module online report
REPORT_SHUTDWN - Module is shutting down, if interface shuts down, modules should prepare for power down.
REPORT_ERROR - Module error report
REPORT_STATUS_POS-Module status report and position, sent at end of requested run and possibly at end any move (joystick etc).
REPORT_STATUS_MTR - Module system status report -
REPORT_ASCII_MSG - Module reports ascii message, mostly for debug purposes currently
REPORT_PWM_POS - Module reports pwm and position
REPORT_SIGNALS_POS - Module reports signals and position
REPORT_SETUP_MTR - Module reports setup values
REPORT_DIAGNOSTIC – for an future reports
REPORT_QUEUE_POS - Module reports queue status and position

Report Configuration: To request a one time report the user would send a SET_LONG_DATA/ REQUEST_REPORT command where the data is the command that is being requested. The report should be transmitted as a response. To configure a report to be transmitted on a timed interval a SET_LONG_DATA/ STATUS_CONFIG command needs to be sent

REPORT_CONFIG:
This is an index into a SET_LONG_DATA/GET_LONG_DATA command and sets up the status report

transmission. The reports may configured to be sent at time intervals, at time intervals during all moves or at timed intervals during requested moves only depending on the four bytes of data. The high order two bytes are the value of the timed interval in milliseconds. The low order byte is the setup byte (definition to follow). The second byte is the requested command. Currently only one command can be requested as an asynchronous report.

STATUS_CONFIG – 4bytes of data to configure report transmission

High order Time in ms.	Low order Time in ms.	Report Command	Report Parameters

Bit#	Label	Description
	REPORT_CONFIG	Transmits Report Setup parameters
0	ALL_MOVES	sent at time int. during all moves.
1	REQUESTED_MOVES	sent at time int. during only requested moves.
2	END_OF_RUN	report is sent at end of run.
3	EVERY_TIME_INT	report is sent based on time interval.
4	START_MOVE	report is sent based at start of move (could be only requested moves).
5	ENDLIMITS	report is sent at endlimit reached.
6	ALL_STOPS	report is sent at all stops.
7	IF_CHANGED	Only set if different from last report sent.

If the value of the report parameter is 0xc1 -- then it will transmit at the interval time during all moves when there has been a change in value and at the endlimits. So the program will interpret combinations of this logic.

REPORTS PARAMETERS:

REPORT_SIGNALS_POS – Each bit in this report represents a signal. These are signal events that may be related to position The first two bytes (sometimes used as the index) will be used to store the signals. The last 4 bytes contains the position. This report is either by a request or set as timed interval report. This report is universal to all modules, but some signals are not applicable for the module.

	SIGNALS	
0	LIMIT_CW	Set when CW limit is active
1	LIMIT_CCW	Set when CCW limit is active
2	LIMIT_HOME	Set when Home limit is active
3	LIMIT_PRE	Set when PRE limit is active
4	SOFT_LOW	Set when soft limits low active
5	SOFT_HIGH	Set when soft limits high active
6	SYNC_INPUT	Set when sync input is active
7	SYNC_OUTPUT	Set when sync output is active – reset on next move
8	SLEW_BUTTON	Set when button on joystick is depressed
9	ADU_PROG	Low when programming the adu from external rs 232
10-15	reserved	

REPORT_STATUS_POS – This report contains 2 bytes of motor move status and 4 bytes of motor position. This report is sent at the end of each requested move regardless of anything else. Bits marked with a '*' are special bits (also called sticky bits) that describe the quality of a user initialized move. These bits

are CLEARED on the start of a user move and SET on the completion of the user move. These bits are only valid when the MOVE_REQUESTED bit is set. Here 'user move' referrers to a user initialized move including all MOTOR_ACTION commands and external generated move such as move filter wheel switch. But not including the joystick or trackball moves. Once these bits are set they will not change until the next user move is initiated. For complex moves (e.g. find center position) these bits should only be set at the end of the completed move.

Bit#	Label	Description
	MOTOR_DONE	
4Bytes	MOTOR_POSITION	Set whenever the current motor position changes. Including joystick and trackball moves.
0	MOVE_REQUESTED	Set whenever the motor is moving due to a command. Excludes joystick and trackball moves. For complex move commands, such as GOTO_ENDLIMIT or CENTER_HOME, this busy bit must remain constantly busy thought the move. Cleared when motor is idle. Will remain on throughout move.
1	MOVE_DONE*	Set on at the end of a motor position move. Cleared on the start of a motor move command. This bit is set weather or not the motor has reached its intended final position. The user must check the remaining bits to check the quality of the move.
2	TARGET_REACHED*	Set when the motor move is done and is within the activation distance. Cleared on the start of a motor move command.
3	USER_STOP*	Set if the user terminated the current move with a stop or halt command. Cleared on the start of a motor move command.
4	INVALID_PAR_STOP*	Set if motor position move was terminated by a parameter error or motor is off. Cleared on the start of a motor move command. A parameter error is invalid move parameter such as a velocity or acceleration value of zero.
5	MOVE_STOP_CW*	Set if the motor move was terminated by the hardware CW limit. Cleared on the start of a motor move command.
6	MOVE_STOP_CCW*	Set if the motor move was terminated by the hardware CCW limit. Cleared on the start of a motor move command.
7	STOP_SOFT_LOW*	Set if the motor move was terminated by the software Low limit. Cleared on the start of a motor move command.
8	STOP_SOFT_HIGH*	Set if the motor move was terminated by the software High limit. Cleared on the start of a motor move command.
9	STOP_STALLED*	Set if the motor move was terminated by a motor crash/stall. Cleared on the start of a motor move command. This bit is set when the actual position exceeds the command position by a preset value. See MOTOR_MAX_PE command.
10	STOP_CURRENT*	Set if the motor move was terminated by an over current condition. Cleared on the start of a motor move command. See MOTOR_MAX_CURRENT
11	STOP_TEMP*	Set if the motor move was terminated by an over temp condition. Cleared on the start of a motor move command. See MOTOR_MAX_TEMP.
12	STOP_HOME_FOUND*	Set if the motor move was terminated by the software Home limit. Cleared on the start of a motor move command.
13	STOP_CW	Stop from CW endlimit hit
14	STOP_CCW	Stop from CCW endlimit hit
15	MOVE_FROM _QUEUE	Move Reuuest from ACTION_QUEUE

REPORT_STATUS_MTR – This report is only sent as a response to a request or through COINFIG_STATUS command. This report contains six bytes of generic status information. The two bytes (often the index bytes) are the system status. The high nibble of the data is the motor_moving status and the low nibble is the current setup status.

Bit#	Label	Description
	SYSTEM	
0	BUSY	This is the system busy bit and is set when the module cannot receive communication. Generally this bit is set when the system is writing to flash, or busy with a complex operation. Users should wait till this bit clears before continuing with communications. This bit should only bit set when absolutely necessary. This bit does not describe the motor move busy status; see REPORT_STATUS_POS bit MOVE_BUSY for motor busy status.
1	BUSY_FLASH	Set when system is busy writing to flash. During this time communications with the module will be disabled
2	MESSAGE_PENDING	Set when a module has an ASCII message pending. See command GET_TEXT_MESSAGE
3	BOOT_RUNNING	Currently running boot code awaiting download of application code
4	ADU_NOT_PRGMD	The adu is not fully programmed for this motor
5	PROG_ADU	High when programming Adu
6-15	Reserved	
	MOTOR_MOVING	
0	MTR_RUNNING	Set when motor is running regardless of from joystick, track , command or servo
1	MOVING_CW	Set when motor is moving in the CW direction
2	RAMPUP	Set when motor is ramp up (accelerating).
3	RAMPING	Set when motor is ramping.
4	DIRECTION	Set when motor is moving in the positive direction.
5	HOME_SEARCH	Set when module is searching for home or end limit
6	BOOST	Set when motor boost (high power) is on
7	SERVOALIGN	Set when motor is servo aligning when idle.
8	ENDLIMIT_SEARCH	Set when motor is doing an endlimit search
9-15	Reserved	Reserved
16	SHUTTER_1	Set when shutter 1 is open.
17	SHUTTER_2	Set when shutter 2 is open.
18	SHUTTER_3	Set when shutter 3 is open.
19	SHUTTER_4	Set when shutter 4 is open.
20	EXPOSURE_1	Set during a timed exposure of shutter 1
21	EXPOSURE_2	Set during a timed exposure of shutter 2
22	EXPOSURE_3	Set during a timed exposure of shutter 3
23	EXPOSURE_4	Set during a timed exposure of shutter 4
24-31	spare	

REPORT_SETUP_MTR – This report is only sent as a response to a request or through COINFIG_STATUS command. This report contains six bytes of generic module setup information.

	SETUP	
0	MOTOR_ON	Set when server checking on
1	SERVON	Set when servo checking is on.
2	JOY_ON	Set when joystick enabled
3	TRACK_ON	Set when trackball enabled.
4	ENCODRE_ON	Set when encoder enabled.
5	REV_DIRECTION	Set when Motor Reversed
6	ENCODER_REV	Set when reverse encoder count
7	CW_SIGNAL_ENABLE	Set when clockwise endlimit enabled.
8	CCW_SIGNAL_ENABLE	Set when counter clockwise endlimit enabled.
9	HOME_ENABLE	Set when joystick enabled
10	PRE_LIMIT_ENABLE	Set when trackball enabled.
11	SOFT_LOW_ ENABLE	Set when encoder enabled.
12	SOFT_HIGH_ ENABLE	Set when Motor Reversed
13	SYNCH_IN_ENABLE	Set when reverse encoder count
14	SYNCH_OUT_ENABLE	Set when clockwise endlimit enabled.
15	ENDLIMIT_SIGNAL_LOW	Set when counter clockwise endlimit enabled.

REPORT_ERROR – This report is sent on any error. The error report value is clear when the report is sent. The error report is 4 bytes in length. Each bit in this report represents an error. The first two bytes (sometimes used as the index) will be used to store the command index that caused the error – if applicable and possible. Because many commands could be sent with out direct response there needs to be a way to tie back problem to originating command.

Bit#	Label	Description
	COMMUNICATION	
0	INVALID_PARM	Set on invalid communication parameter.
1	CAN_ERROR	Set on CAN communication error
2	CAN_IN_FULL	Set on CAN In Buffer Full
3	CAN_OUT_FULL	Set on CAN Out Buffer Full
4	CAN_OUT_TO	Set on CAN Out Time Out
5	INVALID_CMD	Set on invalid, not supported, incorrect syntax or out of parameter range command.
6 -7	Reserved	
	PERIPHERALS	
8	ONE_WIRE_RD	Set on one wire read error
9	ONE_WIRE_WR	Set on one wire write error
10	FLASH_ERROR	Set on Flash error, read, write or erase
11	CORRUPTION	Set on corrupted memory
12	UNAVAILABLE	Set on unavailable resource
13	HARDWARE	Set on hardware error. e.g. PLD, ADUC Step/Dir Chip, etc
14-15	Reserved	

	MOTOR	
16	CANT_MOVE_MOVING	Set when second move request comes in while already moving
17	CONTROL_LOOP	Set on boot up if encoder loop is Unstable, Reversed or Missing
18	MOTOR_STALL	Set on motor stall/crash.
19	MOTOR_CURRENT	Set on motor over current
20	MOTOR_TEMP	Set on motor over temperature
21-32	Reserved	

STATUS_5000 – This status commands has been added to support the older MAC5000 system command structure. Not recommend for new designs. It can be requested at any time. It is not sent asynchronously.

Status 5000 – for stepper motors (6056 and 6054)

Bit#	Label	Description
	Mimics_Cmd126	
0	MTR_BUSY	Set when motor is running or flash is being written too.
1	SERVOALIGN	Set when motor is servo aligning when idle.
2	MOTOR_ON	motor phases are turned on
3	JOYSTICK_ON	Set when joystick enabled
4	RAMPING	Set when motor is ramping.
5	RAMPING_UP	Set when motor is ramping up.
6	CW_SWITCH	Soft copy of the cw end limit switch
7	CCW_SWITCH	Soft copy of the ccw end limit switch
	Mincis_Cmd136	Read Move Status Byte
8	MOVE_COMPLETE	Set on completion of move. Validates rest of information in this byte. Set when the move is completed. When this bit is set, this register will indicate the status of the last move made. Cleared (0)= Moving. Set (1) = Move Done, Not Busy.
9	NORMAL_COMPLETE	Normal end of run – Set when the move has been completed normally. Normally means that the move has ramped up and down as defined by the dsp parameters. It does not necessarily mean that the motor is at the target position. How close this poition is depends on how well the system is tuned. With a rotate move, normally means the rotate move was terminated with a rotate zero command.
10	LESS_TARGET	Set if completed move position is within target +- servo act. Distance This bit is not used with the rotate command.
11	STOP_BY_USER	The motor was stopped by user stop command.
12	STALLED	Motor has stalled, crashed, over temp or over current found
13	INVALID_PARAM	An invalid parameter was sent or motor power off or system busy
14	ENDLIMIT_CW	Set if move terminated by cw endlimit
15	ENDLIMIT_CCW	Set if move terminated by ccw endlimit
		Note: if the capture home is enabled and the home limit is found. Both bits 6 and 7 will be set.
	Mimics_Cmd128	
16	MOTOR_STUCK	Motor stalled
17	SERVO_ON	Servo on
18	EEPROM_EXISTS	Eeprom exists – no longer valid
19	Reserved	
20	Reserved	
21	Reserved	
22	Reserved	
23	Reserved	
	Mimics_Cmd155	
24	OVER_CURRENT	Set on over amp condition
25	RX_BUFFER_FULL	Set on receive buffer full.
26	Reserved	
27	MATH_ERROR	Set on calculation error, range or floating point error.
28	XICOR_ERROR	Not used
29	UNKNOW_COMAMND	Set on unknow or invalid command
30	CONTROL_LOOP	Set on missing, reversed or unstable encoder loop.

31	Reserved	

Status 5000 for Filter Wheels (6081 and 6080)

Bit#	Label	Description
	Mimics_Cmd 115	
0	Timer 1 Status	Exposure timer 1 enabled
1	Timer 1 Status	Exposure timer 2 enabled
2	Shutter 1 Status	Shutter 1 1=open 0 = closed
3	Shutter 2 Status	Shutter 2 1=open 0 = closed
4	Shutter 3 Status	Shutter 3 1=open 0 = closed
5	Roll Over Warning	Not used in 6000
6		
7		