NexStar AUX Command Set

Revision History

Issue	Notes
1.0	Initial release, February 2003

Contents

About this document

This document describes the packet format and protocols used within the NexStar GPS telescopes. I've called these commands the AUX command set. Commands specific to the hand controller are not covered by this document. The AUX command set allows you to:

- use the PC port to control the telescope
- access information like GPS time/site via HC serial port or PC port
- control all aspects of MC operations, like leveling, movement and positioning, backlash settings, etc.
- perform firmware upgrades of the MC
- design other devices that plug into the AUX port that can be accessed from the HC serial or PC port

Contents

This document contains the following chapters:

Disclaimer

This document is the work of the author. No guarantee is made whatsoever about any information in this document. Using it may cause your telescope to melt into a pile of carbon and silicon. Use it at your own risk.

Although this document describes communications used within Celestron's product, Celestron has not endorsed this information and hence it may change or become invalid at any time.

If anyone finds errors, omissions, or additional information pertinent to this document, please send them to me so it can be included.

About the author

Andre Paquette is an engineering graduate and low level software designer employed in the telecommunications industry in Ottawa, Ontario. He owns a Nexstar GPS 8" telescope. Andre has been an amateur astronomer since childhood and received his first telescope at age 10. His astronomy pictures (such as they are) can be found at http://www.paquettefamily.ca/astro.

Contact information

Email Andre Paquette, andre@paquettefamily.ca

URLs This document refers to some programs by others. Links to these resources can be found at:

http://www.paquettefamily.ca/nexstar/

Acknowledgements

A number of individuals have contributed to this project including:

- **Ray St. Denis** — for help with discovering the various message types, and for providing input and corrections. Ray has put together a great demo that takes advantage of the AUX command set.
- **Mike Zeidler** — for help with editing, defining what's in and what's out, and providing access to equipment.
- **Stacey Sheldon** — for helping out with (and teaching me) all things PIC related and with the decoding of the firmware upgrade protocol.
- **Carlene Paquette** — for editing, formatting, and providing input to this document. But mostly for the patience.

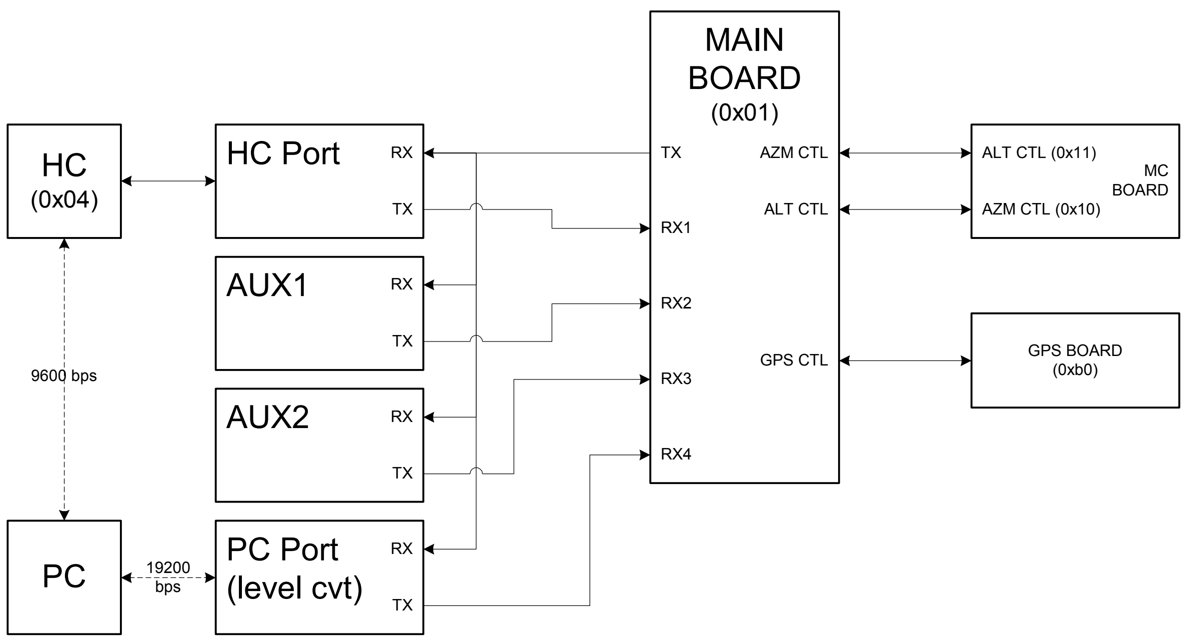
CHAPTER 1

Introduction

To date, the means of accessing the NexStar telescopes with a computer has been via a limited set of commands on the HC serial port. These commands are interpreted by the hand controller and then sent to the appropriate modules on the telescope to obtain the desired effect. The HC command set is not discussed within this document.

Figure 1 shows how the various modules are connected in the NexStar telescope. These modules communicate via serial data, and each module has its own set of commands that it issues and to which it responds. These commands, called the AUX command set within this document, represent the complete functionality of the telescope and are accessible from the AUX ports, the PC port, and the HC serial port.

Figure 1: Interconnection of devices in NexStar telescopes



This document enumerates the AUX commands and describes the packet format and protocols used to communicate between the devices. It is hoped that this information allows people to do new and exciting things with their NexStar telescopes and that interest in these telescopes will increase due to the knowledge of this very flexible and powerful design.

A few obvious things that these commands offer include:

- control of the telescope through the PC port
- access to information such as GPS time/site
- configuration of any aspect of telescope operation
- ability to upgrade firmware
- ability to design devices that make use of the AUX port

Source of information

This information was obtained in part by monitoring packets sent to the PC port from the telescope while operations were performed from the hand controller. The main board echoes every command sent from the HC, so it is possible to see both messages sent and received.

Other information was obtained by analyzing the firmware within the MC and GPS modules.

Warning: Some of the info here is conjecture and based only on a small number of examples of HC<->telescope interaction. There are probably commands available to erase and reprogram the flash on every device. If you try experiments with these program messages or message types that aren't listed, be careful. For example, sending one message of each type to a device would almost certainly cause it to get into a bad state.

Conventions

Packets

Throughout this document, packets and messages are described as sequences of hexadecimal (base 16) bytes. The prefix “0x” is used to indicate that a value is in hexadecimal. For instance, 0x50 is equal to 80 in decimal and the ASCII character ‘P’. Many of the bytes in the packets are not printable characters and can’t easily be entered from a keyboard into a terminal emulator.

Parameters

Parameters are specified with angled brackets around the name of the fields. For example, <msgLen> indicates that a single byte needs to be inserted where the value is the length of the message.

Commands

Each message type is given a label to indicate the purpose of the command and to make it easier to refer to the command. Command names are displayed in a bold font. For example, **`GPS_IS_LINKED`** is a label that represents a GPS command.

Acronyms

Table 1 summarizes the acronyms used in this document.

Table 1: Acronyms

Term	Definition
ALT	Altitude
ASCII	American Standard Code for Information Interchange
ASYNC	Asynchronous serial communications
AUX	Auxiliary
AZM	Azimuth
CTS	Clear To Send (serial handshaking signal)
DEC	Declination
EQ	Equatorial
GND	Ground
GPS	Global Positioning System
HC	Hand Control
LSB	Least Significant Byte/Bit
MC	Motor Controller
MSB	Most Significant Byte/Bit
OTA	Optical Tube Assembly
PEC	Periodic Error Correction
PIC	A family of microcontrollers offered by Microchip. These devices are used throughout Celestron's Nexstar GPS telescopes.
RA	Right Ascension
RS232	Serial interface standard
RTS	Request To Send (serial handshaking signal)
TTL	Transistor-Transistor Logic

CHAPTER 2

AUX packet format

Packet format

Data is sent from one device to another in packets. The format of these packets is as follows:

```
<preamble=0x3b>
<packet len>
<source device #>
<destination device #>
<message ID>
<message data bytes>
<checksum byte>
```

The checksum is computed by summing the bytes starting from the byte after the preamble and ending with the byte before the checksum and taking the LSB of the two's complement. An example of a C program that calculates message checksums is available http://www.paquettefamily.ca/nexstar/examples/calc_cksum.c.

When the HC sends a packet, the main/interconnect board responds with an echo of the same packet that gets sent to all the connected devices. If a device recognizes the destination #, and the command is valid for that device, it will send a packet in response.

Note: MC firmware versions 4.1 and 4.3 respond to packets with a destination # equal to the source # of the original packet. However, firmware version 3.0 responds with a constant destination # of 0x04 (the HC). Other firmware versions are untested at this time.

Raw packet data example

Here is an example of the raw packet data. In this case, the HC is asking the MC processors for their firmware version.

```
# 3 msg bytes, from HC to AZM device, msg ID = 0xfe
HC: <0x3b 0x03 0x04 0x10 0xfe 0xeb>

# main/interconnect now echoes the message we just sent
Main: <0x3b 0x03 0x04 0x10 0xfe 0xeb>

# 5 msg bytes, from AZM to HC, msg ID = 0xfe, msg data = 0x04 0x03
(the firmware version)
AZM: <0x3b 0x05 0x10 0x04 0xfe 0x04 0x03 0xe2>
```

```
# same sequence as above repeated for the ALT destination device
HC: <0x3b 0x03 0x04 0x11 0xfe 0xea>
Main: <0x3b 0x03 0x04 0x11 0xfe 0xea>
ALT: <0x3b 0x05 0x11 0x04 0xfe 0x04 0x03 0xe1>
```

Checksum example

As an example of the checksum, here is the first HC message again:

```
<0x3b 0x03 0x04 0x10 0xfe 0xeb>
```

The checksum can be calculated as follows:

```
0x03+0x04+0x10+0xfe=0x115     # sum the bytes between preamble
                                and checksum

-0x115=0xfeeb                 # take the two's complement of
                                this value

LSB(0xfeeb)=0xeb              # keep only the lower 8 bits
```

Known device IDs

Every device is assigned its own device ID. These IDs are used to address messages to the appropriate destination. The source # and destination # fields from the packet format are set to the appropriate device IDs when messages are sent.

Table 2 summarizes the known device IDs.

Table 2: Known device IDs

Hex ID	Decimal ID	Device name
0x01	1	Main / Interconnection Board
0x04	4	`Hand controller` (HC)
0x10	16	AZM controller on MC board
0x11	17	ALT controller on MC board
0xb0	176	GPS / Compass interface

Some of these device IDs can be seen from the hand controller when it has problems with communications. For instance, the HC errors **`No response 16`** and **`No response 17`** errors that sometimes occur are due to the hand controller not getting a response from the motor control.

Message IDs

The message IDs are unique to the destination device type. For example, the same message ID may be used for one thing on one device and for another thing on a different device.

The list of known message IDs are recorded in tables organized by device as follows:

- Table 3, "Motor control board ALT/AZM messages" on page 13
- Table 4, "GPS/compass messages" on page 25
- Table 5, "Main/interconnect board messages" on page 31

Warning: Message IDs not listed in Table 3, Table 4 and Table 5 may cause damage to the part and some of the listed message types can erase the firmware. Be very cautious when experimenting.

Message format

From this point forward, messages will be displayed in a format where the preamble and checksum are omitted and the names for the device and message IDs will be used instead of the numeric value. Also, the echoed messages from the main/interconnect board will be omitted. For instance, the version request/response message from AZM to HC would look like this:

```
HC->AZM (MC_GET_VER)
AZM->HC (MC_GET_VER, 0x04, 0x03)
```

CHAPTER 3

Interfacing

This chapter describes the various ways of interfacing with the telescope in order to access the AUX command set.

It contains the following sections:

- PC port access
- HC serial access
- AUX port access
- Robust Serial Communications

Note: Schematics are available for the main board, GPS board, and MC board are available online at: http://groups.yahoo.com/group/NexStarGPS/files/n11GPSELECTRONICS.zip.

Note: You must be a member of the NexStar GPS Yahoo group to access this file.

PC port access

The AUX command set can be directly accessed by connecting a PC serial port to the PC port of the telescope. The format is asynchronous serial data with RS232 levels at 19200 bps, no parity, 1 stop bit, and hardware (RTS/CTS) flow control. The appropriate cable can be made following instructions at: http://www.angelfire.com/ns/nexstar/PCControl.htm#ProgrammingCable.

Note: Although it is tempting to use CAT5 (ethernet) cable to make the interface cable, I recommend staying away from any cable that has twisted pairs. Use cables without twists to ensure that no noise is added by twisting the wrong pairs together.

HC serial access

The AUX command set can be accessed via the HC serial port. Interfacing to the serial port is well understood: it uses a 9600 bps, no parity, 1 stop bit, and no flow control. The pinout for the HC serial port can be found within the NexStar user's guide.

There are a small set of published commands for the hand controller. The 0x50 command ID is documented at:
ftp://www.grcooperjr.com/pub/nsol/NewNexStarGPSCommands.zip
as being for configuring the tracking rate. However, this command is actually a means of sending arbitrary AUX commands through the HC serial port to the telescope. There is also a way of receiving responses to those commands, so just about anything than can be done through the PC port can also be done through the HC serial port. The format of these commands is a fixed 8-byte message.

Command format

```
0x50 <msgLen> <destId> <msgId> <data1> <data2> <data3>
<responseBytes>
```

where:

- <msgLen> is the number of bytes in the message. This count includes <msgId> and any valid data bytes.
- <destId> is the destination # from the AUX command set as described in "AUX packet format" on page 5.
- <msgId> is the message ID from the AUX command set as described in "AUX packet format" on page 5.
- <data1-3> are the data bytes of the message. If a data byte is unused it must be set to a value of zero (and not omitted from the message).
- <responseBytes> is the number of bytes to echo back as a response.

Note: If there are more response bytes than specified in the <responseBytes> parameter, the data will be truncated. If <responseBytes> is larger than the number of bytes received, you will receive garbage in the extra bytes.

Command response

After the command is executed, you will get <responseBytes>+1 bytes back from the HC serial interface. The final byte is 0x23, or '#' to indicate that the command is complete.

Example

This is an example of a request and response from an MC firmware version command:

```
# send MC_GET_VER command to device 0x11, expecting 2 bytes back
PC->HC (0x50 0x01 0x11 0xfe 0x00 0x00 0x00 0x02)
HC->PC (0x04 0x03 0x23)
```

AUX port access

External devices (such as the hand controller) can access the AUX command set directly by connecting to the AUX port or the HC port of the telescope. The format is asynchronous serial data with TTL levels (0 to +5V) at 19200 bps, no parity, 1 stop bit, and hardware (RTS/CTS) flow control. The pinout for this port is as follows:

```
pin1 = CTS
pin2 = data from main/interconnect (0-5V)
pin3 = +12V
pin4 = data from device (0-5V)
pin5 = GND
pin6 = RTS
```

Devices can also power themselves from this interface, which is an added bonus.

Caution: Any device connecting to this interface will be limited to some maximum current draw.

Robust Serial Communications

Dealing with lost and corrupted packets is a requirement of every communications protocol. Depending on where you are interfacing (the HC serial port or the PC/AUX port) there are different issues around providing robust serial communications. This section discusses these issues.

PC port and AUX port

There are many points in the system where a packet can be lost or corrupted:

- when sending a packet from you to the main board,
- when the main board echoes the packet from you to the devices,
- when the device sends a response to the main board, and
- when the main board echoes that response so that you can receive it.

Although it is tempting to monitor for the echoed response to a message you send in order to determine if it got sent, I'd suggest avoiding this. For example, the echoed response may have been corrupted on its way to you but not to the destination device, in which case a response will come even though you've given up. Another reason for doing this is that some messages, like the firmware upgrade message, are not echoed by the main board.

The best scheme is simply to wait for the expected response packet and ignore the echo. If you receive any packet with your source # within some set period of time, then you should verify the packet and only retransmit the request if the checksum has failed or the response does not make sense. If you receive no response within the set period of time, then also retransmit the packet. Set a maximum retransmit count and give up if you get no response after some number of attempts.

If you choose a source # with the same value as another device (like the HC) then you may run into problems where you accept a response to messages from the other device. But this would probably only happen during timeouts and for the most part using the same source # as the HC seems to work. However the only reason for using the same source # as the HC is if you were using MC 3.0, which sends all responses to device 0x04. Source # 0x03 seems to be a safe choice when sending messages to the MC and GPS unit on the PC and AUX ports.

HC serial port

Accessing the AUX command set via the HC serial port is fairly easy because the HC takes care of all the checksums, timeouts, and retransmits. If the HC is unable to communicate with the device, it will display on the LCD:

```
No response <XXX>
```

where:

- <XXX> is the device # that did not respond. Values of 16/17 are used for the MC, 176 for GPS. See “Known device IDs” on page 6 for more information.

It is still important to use a timeout/retransmit scheme for sending commands to the HC because data may get lost or corrupted on its way to/from the HC. The timeout period should be set fairly high — larger than the HC's timeout, which appears to be several seconds.

Note: You can not perform a firmware upgrade of the MC via the HC due to the 3-byte limit on tx data. (You need 4 bytes to compose a `MC_PROGRAM_DATA` packet.)

CHAPTER 4

Motor control commands

This chapter enumerates the MC commands (see Table 3) and gives examples of their use.

Note: In almost all of the motor controller commands, the ALT and AZM device are interchangeable. So although some of the examples only show the either ALT or AZM device, both are usually applicable.

Caution: Polling the MC board very frequently can cause some operations to fail. For example, if you `goto` a particular position and then send back-to-back messages to ask if the slew is complete, the MC may miss its destination and keep on rotating.

Table 3: Motor control board ALT/AZM messages

ID	Label	Tx data	Response data	Notes
0x01	MC_GET_POSITION	N/A	24 bits	Get position. Position is a signed fraction of a full rotation.
0x02	MC_GOTO_FAST	16/24 bits	Ack	Goto position with rate = 9. Either 16 or 24 bit accuracy. Position is a signed fraction of a full rotation.
0x04	MC_SET_POSITION	24 bits	Ack	Set position. Position is a signed fraction of a full rotation.
0x06	MC_SET_POS_GUIDERATE	16/24 bits	Ack	Set positive guiderate With 24 bits: • value is rate. With 16 bits: • 0xffff = sidereal • 0xfffe = solar • 0xfffd = lunar

Table 3: Motor control board ALT/AZM messages

ID	Label	Tx data	Response data	Notes
0x07	MC_SET_NEG_GUIDERATE	16/24 bits	Ack	Set negative guide rate With 24 bits: • value is rate. With 16 bits: • 0xffff = sidereal • 0xfffe = solar • 0xfffd = lunar.
0x0b	MC_LEVEL_START	N/A	Ack	Start leveling process. Applicable to ALT only.
0x0c	MC_PEC_RECORD_START	N/A	Ack	Start recording PEC data.
0x0d	MC_PEC_PLAYBACK	8 bits	Ack	Start or stop PEC playback, where: • 0x01 = start playback • 0x00 = stop playback
0x10	MC_SET_POS_BACKLASH	8 bits	Ack	Set positive backlash. Valid range = 0-99.
0x11	MC_SET_NEG_BACKLASH	8 bits	Ack	Set negative backlash. Valid range = 0-99.
0x12	MC_LEVEL_DONE	N/A	8 bits	Determines if scope has finished leveling, where: • 0x00 = not done • 0xff = done Applicable to ALT only.
0x13	MC_SLEW_DONE	N/A	8 bits	Check if slew is complete, where: • 0x00 = not done • 0xff = done
0x14	MC_???	N/A	8 bits	Always returns 0xff. Purpose unknown.
0x15	MC_PEC_RECORD_DONE	N/A	8 bits	Determine if PEC record is complete, where: • 0x00 = not done • 0xff = done
0x16	MC_PEC_RECORD_STOP	N/A	N/A	Stops recording PEC data.
0x17	MC_GOTO_SLOW	16/24 bits	Ack	Goto position with slow, variable rate. Either 16 or 24 bit accuracy. Position is a signed fraction of a full rotation.
0x18	MC_AT_INDEX	N/A	8 bits	Determines if axis is at an index marker, where: • 0x00 = not at index • 0xff = at index Applicable to AZM only.
0x19	MC_SEEK_INDEX	N/A	N/A	Seeks to the nearest index marker. Applicable to AZM only.
0x24	MC_MOVE_POS	8 bits	Ack	Move positive (up/right), where: • value = rate(0 - 9) • rate = 0 means stop

Table 3: Motor control board ALT/AZM messages

ID	Label	Tx data	Response data	Notes
0x25	MC_MOVE_NEG	8 bits	Ack	move positive (down/left) where: • value = rate(0 - 9) • rate = 0 means stop
0x38	MC_ENABLE_CORDWRAP	N/A	N/A	Enables cordwrap feature. Applicable to AZM only.
0x39	MC_DISABLE_CORDWRAP	N/A	N/A	Disables cordwrap feature. Applicable to AZM only.
0x3a	MC_SET_CORDWRAP_POS	24 bits	N/A	Sets cordwrap position. Applicable to AZM only.
0x3b	MC_POLL_CORDWRAP	N/A	8 bits	Determines if cordwrap is enabled or disabled, where: • 0x00 = disabled • 0xff = enabled Applicable to AZM only.
0x3c	MC_GET_CORDWRAP_POS	N/A	24 bits	Gets cordwrap position. Applicable to AZM only.
0x40	MC_GET_POS_BACKLASH	N/A	8 bits	Get positive backlash.
0x41	MC_GET_NEG_BACKLASH	N/A	8 bits	Get negative backlash.
0x46	MC_SET_AUTOGUIDE_RATE	8 bits	Ack	Set autoguide rate (percent of sidereal) Uses the rate percentage formula: • 100 * val / 256
0x47	MC_GET_AUTOGUIDE_RATE	N/A	8 bits	Get autoguide rate (percent of sidereal) Uses the rate percentage formula: • 100 * val / 256
0x81	MC_PROGRAM_ENTER	N/A	8 bits	Enters program mode. See “Firmware Upgrade” on page 23 for details.
0x82	MC_PROGRAM_INIT	N/A	8 bits	Initializes programming sequence. See “Firmware Upgrade” on page 23 for details.
0x83	MC_PROGRAM_DATA	2+2*N bytes	8 bits	Downloads firmware to the device. See “Firmware Upgrade” on page 23 for details.
0x84	MC_PROGRAM_END	N/A	8 bits	Completes programming sequence. See “Firmware Upgrade” on page 23 for details.
0xfc	MC_GET_APPROACH	N/A	8 bits	Get approach, where: • 0 = positive • 1 = negative
0xfd	MC_SET_APPROACH	8 bits	Ack	Set approach, where: • 0 = positive • 1 = negative
0xfe	MC_GET_VER	N/A	16 bits	Get firmware version, where: • MSB is major version # • LSB is minor version #

Querying version

Here's the sequence of packets while retrieving the version number:

```
HC->AZM (MC_GET_VER)
AZM->HC (MC_GET_VER, 0x04, 0x03)
HC->ALT (MC_GET_VER)
ALT->HC (MC_GET_VER, 0x04, 0x03)
```

Note: MC version number was `MC: 4.3 4.3`.

Boot exchange

When the HC is powered up, it exchanges some packets with the MC board:

```
HC->AZM (MC_GET_VER)
AZM->HC (MC_GET_VER, 0x04, 0x03)
HC->AZM (MC_GET_APPROACH)
AZM->HC (MC_GET_APPROACH, 0x00)
HC->ALT (MC_GET_APPROACH)
ALT->HC (MC_GET_APPROACH, 0x01)
```

The HC queries the version of the MC and then sends two more queries to get the approach for each axis.

Querying ALT/AZM

Here's the sequence of packets for retrieving the AZM/ALT:

```
HC->AZM (MC_GET_POSITION)
AZM->HC (MC_GET_POSITION, 0x02, 0x7d, 0xc6)
HC->ALT (MC_GET_POSITION)
ALT->HC (MC_GET_POSITION, 0x00, 0xd1, 0x92)
```

Note: This was for `AZM = 3°,30',12", ALT = 1°,9',4"`.

Note: ALT and AZM are in separate packets because the data is stored in separate controllers on the MC.

Setting ALT/AZM position

The MC controllers deal with position in a relative sense. When north is found or the OTA is levelled, the ALT and AZM position becomes known and these values are sent to the MC board.

The position is sent using the **`MC_SET_POSITION`** command and the data format is the same as the **`MC_GET_POSITION`** command.

Example

```
HC->AZM (MC_SET_POSITION,0xe6, 0xac, 0x7d)
AZM->HC (MC_SET_POSITION)
HC->ALT (MC_SET_POSITION, 0x06, 0xe3, 0x79)
ALT->HC (MC_SET_POSITION)
```

Moving with the arrow keys

When moving the telescope from the HC, a command is issued to start the move and another command issued later to stop it. Specifically, message IDs:

- **0x24** is for *positive* rotations (right/up), and
- **0x25** is for *negative* rotations (left/down).

Note: The message command includes the current rate (0-9) after the msg ID. This rate is the same number entered from the HC keypad.

In addition, it should be noted that:

- **Up/down** commands are issued to the ALT PIC (dest = 0x11), and
- **Right/left** commands are issued to the AZM PIC (dest = 0x10).

Starting an AZM move (right)

```
HC->AZM (MC_MOVE_POS, <rate>)
AZM->HC (MC_MOVE_POS)
```

Note: Starting AZM moves to the left are similar to the above except that the message type of 0x24 is changed to 0x25.

Stopping an AZM move (regardless if start was left/right)

```
HC->AZM (MC_MOVE_POS, 0)
AZM->HC (MC_MOVE_POS)
```

Moving to a specific position

The MC board has no sense of right ascension and declination, only ALT and AZM. When moving to any position, the HC sends a `MC_GOTO_FAST` or `MC_GOTO_SLOW` command in ALT/AZM format. The `MC_SLEW_DONE` command is used to determine if the operation is complete.

The `MC_GOTO_FAST/SLOW` commands can be sent with either 16 or 24 bits of accuracy. For example, you could do a GOTO to 0xfd00 or to 0xfd0080 if you wanted/needed the extra accuracy.

Example

```
HC->AZM (MC_GOTO_FAST, 0x4a, 0xdd, 0xf2)
AZM->HC (MC_GOTO_FAST)
HC->ALT (MC_GOTO_FAST, 0x12, 0xb9, 0x77)
ALT->HC (MC_GOTO_FAST)

HC->AZM (MC_SLEW_DONE)
AZM->HC (MC_SLEW_DONE, 0x00) # not done yet
HC->ALT (MC_SLEW_DONE)
ALT->HC (MC_SLEW_DONE, 0x00) # not done yet

HC->AZM (MC_SLEW_DONE)
AZM->HC (MC_SLEW_DONE, 0xff) # slew complete
HC->ALT (MC_SLEW_DONE)
ALT->HC (MC_SLEW_DONE, 0xff) # slew complete
```

Note: The motion on the axes are independent, so when one axis returns 0xff it is only necessary to continue polling the other axis.

Controlling Backlash

Backlash is controlled via the **MC_GET_POS_BACKLASH**, **MC_SET_POS_BACKLASH**, **MC_GET_NEG_BACKLASH**, and **MC_SET_NEG_BACKLASH** commands.

Get ALT/AZM positive backlash

```
HC->AZM (MC_GET_POS_BACKLASH)
AZM->HC (MC_GET_POS_BACKLASH, 0x00) # currently 0
```

Set ALT/AZM positive backlash

```
HC->AZM (MC_SET_POS_BACKLASH, 0x01) # set to 1
AZM->HC (MC_SET_POS_BACKLASH)
```

Get ALT/AZM negative backlash

```
HC->AZM (MC_GET_NEG_BACKLASH)
AZM->HC (MC_GET_NEG_BACKLASH, 0x00) # currently 0
```

Set ALT/AZM negative backlash

```
HC->AZM (MC_SET_NEG_BACKLASH, 0x01) # set to 1
AZM->HC (MC_SET_NEG_BACKLASH)
```

Controlling GOTO Approach

Positive or negative approach is controlled via the **MC_GET_APPROACH**, and **MC_SET_APPROACH** commands.

Set ALT/AZM approach

```
HC->AZM (MC_SET_APPROACH, 0x00) # 0 = positive, 1 = negative
AZM->HC (MC_SET_APPROACH)
```

Get ALT/AZM approach

```
HC->AZM (MC_GET_APPROACH)
AZM->HC (MC_GET_APPROACH, 0x00) # 0 = positive, 1 = negative
```

Controlling Autoguide Rate

Autoguide rate is configured as a percentage of sidereal rate. The `MC_GET_AUTOGUIDE_RATE` and `MC_SET_AUTOGUIDE_RATE` commands allow you to control this configuration. The percentage returned/set is calculated as:

```
percentage = 100 * value / 256
```

Get ALT/AZM Autoguide rate

```
HC->AZM (MC_GET_AUTOGUIDE_RATE)
AZM->HC (MC_GET_AUTOGUIDE_RATE, 0x80) # 0x80 = 50%
```

Set ALT/AZM Autoguide rate

```
HC->AZM (MC_SET_AUTOGUIDE_RATE, 0x1a) # 0x1a = 10% (value = pct / 100 * 256)
AZM->HC (MC_SET_AUTOGUIDE_RATE)
```

Tracking modes

The HC controls the tracking of the telescope by setting either a positive or negative guide rate for each axis. For EQ North and South, this means sending a single message to the AZM controller to tell it to go forward or backwards. For ALT/AZM tracking, the HC sets an initial tracking rate for each axis and then updates that rate every 30 seconds depending on the current ALT/AZM position retrieved.

Examples

Configure for EQ-North

```
HC->AZM (MC_SET_POS_GUIDERATE, 0x00, 0x00, 0x00) # stop
AZM->HC (MC_SET_POS_GUIDERATE)
HC->ALT (MC_SET_POS_GUIDERATE, 0x00, 0x00, 0x00) # stop
ALT->HC (MC_SET_POS_GUIDERATE)
HC->AZM (MC_SET_POS_GUIDERATE, 0xff, 0xff) # rate for EQ North, sidereal
AZM->HC (MC_SET_POS_GUIDERATE)
```

Configure for EQ-South

```
HC->AZM (MC_SET_POS_GUIDERATE, 0x00, 0x00, 0x00) # stop
AZM->HC (MC_SET_POS_GUIDERATE)
HC->ALT (MC_SET_POS_GUIDERATE, 0x00, 0x00, 0x00) # stop
ALT->HC (MC_SET_POS_GUIDERATE)
HC->AZM (MC_SET_NEG_GUIDERATE, 0xff, 0xff) # rate for EQ South, sidereal
AZM->HC (MC_SET_NEG_GUIDERATE)
```

Configure for ALT/AZM

```
HC->AZM (MC_SET_POS_GUIDERATE, 0x00, 0x00, 0x00) # stop
AZM->HC (MC_SET_POS_GUIDERATE)
HC->ALT (MC_SET_POS_GUIDERATE, 0x00, 0x00, 0x00) # stop
ALT->HC (MC_SET_POS_GUIDERATE)
# wait 30 seconds
HC->AZM (MC_GET_POSITION)
AZM->HC (MC_GET_POSITION, 0x22, 0x12, 0x8e)
HC->ALT (MC_GET_POSITION)
ALT->HC (MC_GET_POSITION, 0x10, 0xfe, 0x06)
HC->AZM (MC_SET_P0S_GUIDERATE, 0x00, 0x1d, 0xef)
AZM->HC (MC_SET_POS_GUIDERATE)
HC->ALT (MC_SET_POS_GUIDERATE, 0x00, 0x1f, 0x90)
ALT->HC (MC_SET_POS_GUIDERATE)
# wait 30 seconds
HC->AZM (MC_GET_POSITION)
AZM->HC (MC_GET_POSITION, 0x22, 0x1d, 0xe8)
HC->ALT (MC_GET_POSITION)
ALT->HC (MC_GET_POSITION, 0x11, 0x09, 0xf7)
HC->AZM (MC_SET_P0S_GUIDERATE, 0x00, 0x1d, 0xe9)
AZM->HC (MC_SET_POS_GUIDERATE)
HC->ALT (MC_SET_POS_GUIDERATE, 0x00, 0x1f, 0x98)
ALT->HC (MC_SET_POS_GUIDERATE)
# wait 30 seconds
HC->AZM (MC_GET_POSITION)
AZM->HC (MC_GET_POSITION, 0x22, 0x28, 0x77)
HC->ALT (MC_GET_POSITION)
ALT->HC (MC_GET_POSITION, 0x11, 0x15, 0x22)
HC->AZM (MC_SET_P0S_GUIDERATE, 0x00, 0x1d, 0xe4)
AZM->HC (MC_SET_POS_GUIDERATE)
HC->ALT (MC_SET_POS_GUIDERATE, 0x00, 0x1f, 0xa0)
ALT->HC (MC_SET_POS_GUIDERATE)
```

Tracking Rate

The tracking rate can be set to one of Sidereal, Solar, or Lunar. When in EQ North/South mode, the tracking rate is set via a special `MC_SET_POS/NEG_GUIDERATE` command as shown in "Tracking modes" on page 19. The values for this command are as follows:

- `0xffff = sidereal`
- `0xfffe = solar`
- `0xfffd = lunar`

This data is sent via a 2-data byte packet instead of the 3-byte packets used to control the rate in ALT/AZM mode.

Note: Although the ALT controller can be configured for various linear tracking rates, it only makes sense to send these commands to the AZM controller.

Leveling the Scope

The **MC_LEVEL_START** and **MC_LEVEL_DONE** commands are used to level the telescope. The HC sends **MC_LEVEL_START** and sends **MC_LEVEL_DONE** commands to poll to see if the process is complete.

Example

```
HC->ALT (MC_LEVEL_START)
ALT->HC (MC_LEVEL_START)

HC->ALT (MC_LEVEL_DONE)
ALT->HC (MC_LEVEL_DONE, 0x00) # level not yet complete

HC->ALT (MC_LEVEL_DONE)
ALT->HC (MC_LEVEL_DONE, 0xff) # level complete
```

Note: The level commands can be sent to the AZM controller but it always returns a value of 0x80 to the **MC_LEVEL_DONE** command.

Note: The telescope initially slews in a positive direction (up) when leveling, so it is beneficial to manually position the tube below the level point before sending the **MC_LEVEL_START** command.

After the MC reports that the level is complete, the HC then moves the OTA a short distance from that point. I suspect that this is part of the level calibration: the MC reports level at a particular position, and then the HC corrects for errors in that position.

PEC Operations

Before PEC operations are performed, the HC instructs the MC to find the index marker on the AZM device. This is done via the **MC_AT_INDEX** and **MC_SEEK_INDEX** commands. Once the AZM is at the index marker, the PEC playback or recording can start. This is controlled via the **MC_PEC_PLAYBACK**, **MC_PEC_RECORD_START**, **MC_PEC_RECORD_STOP**, and **MC_PEC_RECORD_DONE** commands.

Note: The MC must be configured for EQ North/South tracking in order for these commands to operate as expected.

PEC record example

Here is an example sequence for recording the PEC:

```
HC->AZM (MC_AT_INDEX)
AZM->HC (MC_AT_INDEX, 0x00)        # not at the index marker
HC->AZM (MC_SEEK_INDEX)
AZM->HC (MC_SEEK_INDEX)            # look for the index
HC->AZM (MC_AT_INDEX)
AZM->HC (MC_AT_INDEX, 0x00)        # found it yet?  No.
HC->AZM (MC_AT_INDEX)
AZM->HC (MC_AT_INDEX, 0xff)        # found it yet?  Yes.
HC->AZM (MC_PEC_RECORD_START)
AZM->HC (MC_PEC_RECORD_START)      # start recording
HC->AZM (MC_PEC_RECORD_DONE)
AZM->HC (MC_PEC_RECORD_DONE, 0x00) # are we done yet?  No.
HC->AZM (MC_PEC_RECORD_DONE)
```

... 8 minutes later...

```
AZM->HC (MC_PEC_RECORD_DONE, 0xff) # are we done yet?  Yes.
```

If the user had hit abort before the MC finished, the HC would send:

```
HC->AZM (MC_PEC_RECORD_STOP)
AZM->HC (MC_PEC_RECORD_STOP)
```

PEC playback example

Here is an example for playback of PEC data:

```
HC->AZM (MC_AT_INDEX)
AZM->HC (MC_AT_INDEX, 0xff)        # already at index, so no need to seek
HC->AZM (MC_PEC_PLAYBACK, 0x01)
AZM->HC (MC_PEC_PLAYBACK)          # start PEC playback
```

... when finished ...

```
HC->AZM (MC_PEC_PLAYBACK, 0x00)
AZM->HC (MC_PEC_PLAYBACK)          # stop PEC playback
```

Note: Although the `MC_PEC` commands can be sent to both ALT and AZM devices, use with the ALT device doesn't make sense both operationally and because the `MC_INDEX` commands only work on the AZM device.

Controlling Cordwrap

Cordwrap is controlled via the `MC_ENABLE_CORDWRAP`, `MC_DISABLE_CORDWRAP`, `MC_SET_CORDWRAP_POS`, `MC_GET_CORDWRAP_POS`, and `MC_POLL_CORDWRAP` commands.

To enable cordwrap mode

```
HC->AZM (MC_ENABLE_CORDWRAP)
AZM->HC (MC_ENABLE_CORDWRAP)
```

To disable cordwrap mode

```
HC->AZM (MC_DISABLE_CORDWRAP)
AZM->HC (MC_DISABLE_CORDWRAP)
```

To set the AZM position for cordwrap

```
HC->AZM (MC_SET_CORDWRAP_POS, 0x01, 0x02, 0x03)  # set cordwrap position to
0x010203
AZM->HC (MC_SET_CORDWRAP_POS)
```

Note: When setting the cordwrap, the position should be set to the current AZM position + 180° mod 360°.

To get the current AZM position for cordwrap

```
HC->AZM (MC_GET_CORDWRAP_POS)
AZM->HC (MC_GET_CORDWRAP_POS, 0x01, 0x02, 0x03) # get cordwrap position: 0x010203
```

Firmware Upgrade

Warning: Executing the `MC_PROGRAM_ENTER` command places the MC into a programming mode. This command prevents normal MC commands from being executed. Do not enter this command unless you know what you are doing.

See the Firmware upgrades chapter, which describe the `MC_PROGRAM` commands in detail.

CHAPTER 5

GPS commands

This chapter enumerates the GPS commands (see Table 4) and gives examples of their use.

Table 4: GPS/compass messages

ID	Label	Tx data	Response data	Notes
0x01	GPS_GET_LAT	N/A	24 bits	Gets latitudinal position. Position is a signed fraction of a full rotation.
0x02	GPS_GET_LONG	N/A	24 bits	Gets longitudinal position. Position is a signed fraction of a full rotation.
0x03	GPS_GET_DATE	N/A	16 bits	Get date in GMT, where: • first byte is month • second byte is day
0x04	GPS_GET_YEAR	N/A	16 bits	Get year in GMT
0x07	GPS_GET_SAT_INFO	N/A	16 bits	Gets satellite info, where: • first byte is number of visible satellites • second byte is number of satellites being tracked
0x08	GPS_GET_RCVR_STATUS	N/A	16 bits	Gets receiver status. See "Receiver status" on page 29 for format description.
0x33	GPS_GET_TIME	N/A	24 bits	Get time in GMT, where: • first byte is hours • second is minutes • third is seconds
0x36	GPS_TIME_VALID	N/A	8 bits	Poll to see if time is valid, where: • 0 = no/false • 1 = yes/true
0x37	GPS_LINKED	N/A	8 bits	Poll to see if GPS is linked, where: • 0 = no • 1 = yes
0x55	GPS_GET_HW_VER	N/A	8 bits	Returns 0xAB. Believed to be the HW version of the Motorola GPS module.

Table 4: GPS/compass messages

ID	Label	Tx data	Response data	Notes
0xa0	GPS_GET_COMPASS	N/A	8 bits	Get compass heading, where: • N = 0x0b • NE = 0x09 • E = 0x0d • SE = 0x0c • S = 0x0e • SW = 0x06 • W = 0x07 • NW = 0x03
0xfe	GPS_GET_VER	N/A	16 bits	Get firmware version, where: • MSB is major version # • LSB is minor version #

View time/site

Note: The year, month/day, and time are all in GMT.

Here are the packet sequences that occur when viewing time/site info:

Check time

```
# is time valid
HC->GPS (GPS_TIME_VALID)
GPS->HC (GPS_TIME_VALID, 0x01) # yes, time is valid
```

Note: The checks for time (0x33) and GPS validity (0x36).
GPS validity returns:
- 0 for no/false and
- 1 for yes/true.

Year

```
# get the year
HC->GPS (GPS_GET_YEAR)
GPS->HC (GPS_GET_YEAR, 0x07, 0xd3) # 2003
```

Note: The year is a 16 bit integer. For example, 0x7d3 = 2003.

Month/day

```
# get the month/day
HC->GPS (GPS_GET_DATE)
GPS->HC (GPS_GET_DATE, 0x01, 0x10) # Jan 16
```

Note: The month/day are in two 8-bit fields.

Time

```
# get the time
HC->GPS (GPS_GET_TIME)
GPS->HC (GPS_GET_TIME, 0x11, 0x2b, 0x16) # 17h43m22s GMT
```

Note: The time is in three 8-bit fields.

Check GPS link

```
# is GPS linked?
HC->GPS (GPS_LINKED)
GPS->HC (GPS_LINKED, 0x00) # GPS searching
HC->GPS (GPS_LINKED)
GPS->HC (GPS_LINKED, 0x01) # GPS is linked
```

Longitude

```
# get the longitude
HC->GPS (GPS_GET_LONG)
GPS->HC (GPS_GET_LONG, 0xca, 0x06, 0x00) # -75°54'16.35"
```

Latitude

```
# get the latitude
HC->GPS (GPS_GET_LAT)
GPS->HC (GPS_GET_LAT, 0x20, 0x3e, 0x35) # +45°20'30.17"
```

Note: The latitude and longitude are 24-bit signed integers representing a fraction of 360 degrees. You can find an example C program for converting this to degrees, minutes, and seconds online at: http://www.paquettefamily.ca/nexstar/examples/location.c.

Finding north

The NexStar compass position can be obtained by sending the **GPS_GET_COMPASS** command. Only 8 different values are returned, one for each ordinal point. The return values are:

- N = 0x0b
- NE = 0x09
- E = 0x0d
- SE = 0x0c
- S = 0x0e
- SW = 0x06
- W = 0x07
- NW = 0x03

The sequence to obtain the compass position looks like:

```
HC->GPS (GPS_GET_COMPASS)
GPS->HC (GPS_GET_COMPASS, 0x0b) # north
```

The algorithm that the HC uses to more accurately point the telescope north is to find the AZM position corresponding to the transition points between NE->N and NW->N and then slew to a position half-way between these two points.

Example

```
HC->AZM (MC_SET_POS_GUIDERATE, 0x00, 0x00, 0x00) # stop
AZM->HC (MC_SET_POS_GUIDERATE)
HC->ALT (MC_SET_POS_GUIDERATE, 0x00, 0x00, 0x00) # stop
ALT->HC (MC_SET_POS_GUIDERATE)

HC->GPS (GPS_GET_COMPASS) # where are we pointing?
GPS->HC (GPS_GET_COMPASS, NW)
HC->GPS (GPS_GET_COMPASS)
GPS->HC (GPS_GET_COMPASS, NW)

HC->AZM (MC_MOVE_POS, rate=9) # move towards N
AZM->HC (MC_MOVE_POS)

HC->GPS (GPS_GET_COMPASS) # keep doing this until N appears
GPS->HC (GPS_GET_COMPASS, NW)

HC->GPS (GPS_GET_COMPASS) # found north
GPS->HC (GPS_GET_COMPASS, N)

HC->AZM (MC_MOVE_POS, rate=0) # stop
AZM->HC (MC_MOVE_POS)
HC->AZM (MC_GET_POSITION)
AZM->HC (MC_GET_POSITION, 0x03, 0x5d, 0x1c) # record AZM position

HC->GPS (GPS_GET_COMPASS) # still north?
GPS->HC (GPS_GET_COMPASS, N)
HC->AZM (MC_MOVE_POS, rate=9) # keep moving in same direction
AZM->HC (MC_MOVE_POS)

HC->GPS (GPS_GET_COMPASS) # keep doing this until NE appears
GPS->HC (GPS_GET_COMPASS, N)

HC->GPS (GPS_GET_COMPASS) # found NE
GPS->HC (GPS_GET_COMPASS, NE)

HC->AZM (MC_MOVE_POS, rate=0) # stop
AZM->HC (MC_MOVE_POS)

HC->GPS (GPS_GET_COMPASS) # still NE?
GPS->HC (GPS_GET_COMPASS, NE)
HC->AZM (MC_MOVE_NEG, rate=9) # now move in opposite dir until we pass N
AZM->HC (MC_MOVE_NEG)

HC->GPS (GPS_GET_COMPASS) # found N
GPS->HC (GPS_GET_COMPASS, N)

HC->AZM (MC_MOVE_POS, rate=0) # stop
AZM->HC (MC_MOVE_POS)
HC->AZM (MC_MOVE_POS, rate=1) # start moving slowly forward -- why?
AZM->HC (MC_MOVE_POS)
HC->AZM (MC_GET_POSITION)
AZM->HC (MC_GET_POSITION, 0x26, 0x78, 0xdd) # record AZM position

HC->AZM (MC_GOTO_FAST, 0x13, 0x23, 0xe0) # goto calculated north position
AZM->HC (MC_GOTO_FAST)
HC->AZM (MC_SLEW_DONE)
AZM->HC (MC_SLEW_DONE, 0x00) # wait for slew to complete

HC->AZM (MC_SLEW_DONE)
AZM->HC (MC_SLEW_DONE, 0x01) # slew is complete
```

Satellite info

The **GPS_GET_SAT_INFO** command returns two bytes:

- the number of satellites currently visible to the GPS module
- the number of satellites being tracked

The count of visible satellites seems to take a long time to become valid while the count of tracked satellites is updated very frequently. By blocking the GPS antenna (located in the fork arm with the handle) with your hand, you can watch these values change in response to the changing signal.

Receiver status

The **GPS_GET_RCVR_STATUS** command returns two bytes with information about the status of the GPS receiver. This data is in a bit-mapped format as follows:

```
Bit 15-13:           111 = 3D Fix
                     110 = 2D Fix
                     101 = Propagate Mode
                     100 = Position Hold
                     011 = Acquiring Satellites
                     010 = Bad Geometry
                     001 = Reserved
                     000 = Reserved
Bit 12-11:           Reserved
Bit 10:              Narrow track mode (timing only)
Bit 9:               Fast Acquisition Position
Bit 8:               Filter Reset To Raw GPS Solution
Bit 7:               Cold Start (no almanac out of date or
                     have almanac nut time or position unknown)
Bit 6                Differential Fix
Bit 5:               Position Lock
Bit 4:               Autosurvey Mode
Bit 3:               Insufficient Visible Satellites
Bit 2-1:             Antenna Sense   00 = OK
                                     01 = OC
                                     10 = UC
                                     11 = NV
Bit 0:               Code Location   0 = EXTERNAL
                                     1 = INTERNAL
```

This information can be found in chapter 5 of Motorola's GPS user's guide at: http://www.motorola.com/ies/GPS/products/positioningmodules.html

When the GPS module is searching for satellites, bits 15-13 are set to `Acquiring Satellites`. Once there is enough data for a link, these bits change to `2D Fix`. If there is enough signal and satellites, this values changes to `3D Fix`.

Hardware version

The **GPS_GET_HW_VER** command returns a constant value of 0xab. I believe this value is the hardware version of the Motorola GPS device based on the fact that 0xab appears on the GPS board.

Firmware version

The GPS device firmware version can be retrieved using the same command value as for the MC board. On my telescope, the device returns version 1.0:

```
HC->GPS (GPS_GET_VER)
GPS->HC (GPS_GET_VER, 0x01, 0x00)
```

CHAPTER 6

Main interconnect commands

The main/interconnect board does not appear to terminate or originate any commands in the regular use of the telescope. However, it does respond to some commands summarized in Table 5.

Table 5: Main/interconnect board messages

ID	Label	Tx data	Response data	Notes
0x01	???			Responds to this message with: • dest deviceId = 0x20.
0xfe	MAIN_GET_VER	N/A	16 bits	Get firmware version, where: • MSB is major version # • LSB is minor version #

CHAPTER 7

Firmware upgrades

This chapter discusses the process of upgrading device firmware in the NexStar telescope.

MC firmware upgrade

This section describes how to upgrade the firmware of the motor controller via the AUX command set. It also gives an overview of the PIC processors on the MC board and offers resources describing how to manually program PICs with a programmer, should the need arise.

PIC resources

If you intend to experiment with MC firmware upgrades, I strongly recommend that you purchase a PIC programmer. The programmer allows you to extract firmware from the device and to reprogram it in order to do an upgrade or to recover from a failed attempt at the command-based upgrade process. It also allows you to alter portions of the PIC software that are not accessible via the AUX command set.

Buying a PIC programmer

Programmers are not expensive and can be found at: http://www.phanderson.com/.

PIC development software

The MPLAB software for the programmer can be downloaded for free from Microchip at: http://www.microchip.com/1010/pline/tools/picmicro/devenv/mplabi/mplab5x/index.htm.

PIC processor information

Detailed information on the PIC16F876 processor used in the MC board can be found at Microchip at: http://www.microchip.com/download/lit/pline/picmicro/families/16f87x/30292c.pdf.

MC schematics

Schematics for the MC board can be found at: http://groups.yahoo.com/group/NexStarGPS/files/n11GPSELECTRONICS.zip

Intel Hex format

Information on the Intel Hex format can be found at: http://www.cs.net/lucid/intel.htm

Firmware tested

The process described in this chapter has been tested on MC firmware versions 3.0, 4.1, and 4.3. Successful serial upgrades were done from 3.0 to 4.1, 3.0 to 4.3, and 4.1 to 4.3. Downgrading was also tested within these firmware versions.

It is not possible to upgrade firmware version 2.0 using the AUX command set. In order to upgrade version 2.0, either a PIC programmer must be used or a new MC board obtained from Celestron.

PIC memory layout

The addressing on the MC PIC processors is done on a word basis. Each address refers to a 14-bit value (word) which can hold an opcode or data. When the firmware is extracted to an Intel HEX file using MPLAB, each word is stored as two bytes with the LSB first, then the MSB.

From the perspective of firmware upgrades, the PIC memory is organized into four sections as shown in Table 6:

Table 6: PIC memory map

Address Range	Description	Upgrade Access
`0x0000 - 0x0003`	Jump to boot	Read Only
`0x0004 - 0x1eff`	Main application	Read/Write
`0x1f00 - 0x1ffb`	Boot code	Read Only
`0x1ffc - 0x1fff`	Jump to main	Read/Write

The boot code controls the firmware upgrade process and can not be modified with the AUX command set. Similarly, the initial code which jumps into the boot area can not be altered. A PIC programmer is the only means of altering this code.

The firmware upgrade process allows you to change the main application section and the instructions which jump into the main area after booting has completed.

Programming Sequence

Firmware upgrade is controlled via the **`MC_PROGRAM_ENTER`**, **`MC_PROGRAM_INIT`**, **`MC_PROGRAM_DATA`**, and **`MC_PROGRAM_END`** commands.

The **`MC_PROGRAM_ENTER`** command takes no parameters and places the MC PIC corresponding to the destination # in an upgrade mode. All **`MC_PROGRAM`** commands respond with a source # of 0x1d. Although it is not necessary to send commands from that source, doing so makes the implementation of message handlers a bit easier.

Warning: Do not try to upgrade both PICs at once. After the **`MC_PROGRAM_ENTER`** command is sent during the MC PIC upgrade, the other device will stop responding.

Note: It is recommended that you disconnect the HC before upgrading the firmware in order to prevent messages from the HC from interfering with the upgrade.

When in the programming mode, the PIC will only accept **MC_PROGRAM_INIT**, **MC_PROGRAM_DATA**, and **MC_PROGRAM_END** commands. Once the process is complete, the PIC will reboot with the new firmware. If the MC should happen to reboot in the middle of this process, it will return to programming mode (as if **MC_PROGRAM_ENTER** had been sent) and will only accept the **MC_PROGRAM_INIT**, **MC_PROGRAM_DATA**, and **MC_PROGRAM_END** commands.

MC_PROGRAM_INIT starts off the download process and once this command is sent, only **MC_PROGRAM_DATA** and **MC_PROGRAM_END** commands will be accepted.

After all data has been written, the **MC_PROGRAM_END** command is sent and the PIC reboots, hopefully running the new code.

Caution: Sending **MC_PROGRAM_END** is the last thing that is done before the download is completed. If the memory is in a corrupted or intermediate state when this command is sent, the PIC device will probably become unusable and will require a PIC programmer in order to be reprogrammed.

Data packet format

Data is programmed using the **MC_PROGRAM_DATA** command. The data payload in this packet is formatted as follows:

```
<addrHi><addrLo><dataLo1><dataHi1>,[<dataLo2><dataHi2>, ...,
<dataLoN><dataHiN>]
```

where:

- <addrHi> is the MSB of the start address
- <addrLo> is the LSB of the start address
- <dataLo> is the LSB of the data word
- <dataHi> is the MSB of the data word

A maximum of 10 payload bytes (2 address + 8 data) is allowed in the **MC_PROGRAM_DATA** command. A minimum of 4 payload bytes (2 address + 2 data) is required to compose a valid message.

Note: The **MC_PROGRAM_DATA** command is not echoed by the main/interconnect board as is the case with all the other commands.

Address fields

Valid address values range from 0x0000 - 0x0003 and 0x0004 - 0x1eff. Values of 0x0000 - 0x0003 are special in that they are used to indicate that addresses 0x1ffc - 0x1fff be written. So when writing data with the address fields set to 0x0000 - 0x0003, be sure that the data values correspond to the contents of 0x1ffc - 0x1fff. Values from 0x0004 - 0x1eff are directly mapped onto the same memory addresses. See Table 6 on page 34 for the PIC memory map.

Return Values

All **MC_PROGRAM** commands return a single message as a result. Rather than placing the result in message data, separate message types are used for each return code. Here are the possible message IDs corresponding to return values:

```
0x02 = success
0x03 = program mode entered
0x05 = unexpected msg rxd
0x06 = verify failed
0x07 = checksum failed
0x80 = address range error
```

Example programming sequence

```
PC->AZM (MC_PROGRAM_ENTER)
AZM->PC (0x03)
PC->AZM (MC_PROGRAM_INIT)
AZM->PC (0x02)
# set the return vector data (0x1ffc - 0x1fff)
PC->AZM (MC_PROGRAM_DATA, 0x00,0x00,0x0A,0x20,0xFF,0x3F,0xFF,0x3F,0xFF,0x3F)
AZM->PC (0x02)
# send main program data (0x0004 - 0x1eff)
PC->AZM (MC_PROGRAM_DATA, 0x00,0x04,0x02,0x01,0x04,0x03,0x06,0x05,0x08,0x07)
AZM->PC (0x02)
```

... remaining data packets sent ...

```
PC->AZM (MC_PROGRAM_END)
AZM->PC (0x02)
```

Warning: There appears to be a bug in the main/interconnect board regarding sending **MC_PROGRAM_DATA** commands with 0x3b in the packet data. If the byte following the 0x3b is equal to the number of bytes left in the packet (including itself), then the main/interconnect board will not forward the packet to the MC. A workaround is to increase/decrease the packet length, or otherwise alter the position of the 0x3b within the packet, so that it is accepted and forwarded.

GPS firmware upgrade

Due to the nature of the GPS 1.0 firmware, it is not possible to upgrade the GPS module via the AUX command set. However, the PIC device used in the GPS module is the same as the PICs used for the motor controller and hence the "PIC resources" on page 33 can be used to help examine, upgrade, or alter the GPS PIC code.

CHAPTER 8

Outstanding mysteries

I suspect there will always be a list of things that we don't understand but intend to investigate at some point in the future. Here is a list of such things:

- What additional commands does the main/interconnect board support. What is command 0x01 doing?
- Is "King" a rate option in the `MC_SET_GUIDERATE` commands?
- What is the architecture and command structure for the i-series telescope?
- What is the difference in the set of commands supported by the various MC versions.
- What is the MC message ID 0x14 used for?
- Which HC versions support the 0x50 proxying command?
- What is the firmware upgrade process for the HC and other devices. Does it look like the MC firmware upgrade? Does it use the same commands?