SkyWalker Client Interface Specification and Programming Guide

Last revision date: 23-Dec-2008

This document can be ordered as Astrometric Instruments' part DOC-11

Contents

Version

For SkyWalker version 0.90.000 (or later) firmware. For pre-1.00.000 firmware this specification is subject to drastic, erratic, oscillatory and un-announced change.

Preface

Astrometric Instrument's next generation telescope control system (referred to as Astrometric Telescope Control System version 4 or ATCS4) moves the system's motion control intelligence from PC-based software (i.e. SkyGuide) into hardware-based firmware, either inside new SkyWalkers or inside the S-Box peripheral (which enables ATCS4 functionality for older "dumb" SkyWalkers).

To complete the telescope control system, a user interface to the intelligence within SkyWalker is required. Astrometric Instruments provides a PC-based "Client" program that provides one possible user interface to SkyWalker. This Client program is called Maestro. Maestro is not required to operate SkyWalker. SkyWalker can also be operated stand-alone (i.e. without an attached PC) with one of two handpaddle models as user interface: HP1 or HP2. HP1 is a basic handpaddle that provides keyed motion, change of View Velocity, change of Track rate, Focus/Dome control and access to 3 Quick keys. HP2 provides all the features of HP1 in addition to an 8-line display and full keypad that provides the entire functionality of SkyWalker's *Instrument Display* interface. The *Instrument Display* interface provides a hierarchical menu-based interface to nearly all of SkyWalker's features.

Astrometric Instruments' SkyWalker client, Maestro, exposes SkyWalker's *Instrument Display* interface **and** provides a full Microsoft Windows based graphical user interface to SkyWalker. For those familiar with Astrometric's older SkyGuide software, these two components of Maestro are similar to SkyGuide's *Instrument Display* and SkyGuide's *Console Tabs* GUI interface (i.e. Objects, Status, Actions and Settings tabs, among others).

Maestro communicates to SkyWalker via the Astrometric Telescope Control Language (ATCL). ATCL is typically communicated over serial link (e.g. RS232 or USB) to SkyWalker's Client interface port (labeled "Com"). ATCL provides a very rich syntax for telescope control.

Astrometric Instruments, Inc. encourages independent software developers to provide an ATCL interface to their software. ATCL is an open language. There are no proprietary syntax or commands that are hidden from this standard. All the information necessary to use ATCL is provided in this document. In fact, Astrometric Instruments, Inc. provides an example SkyWalker Client application, Maestro-Lite, which is a stripped-down version of Maestro, to serve as a working example for independent software developers wishing to implement ATCL in their software. Maestro-Lite is written in Microsoft Visual Basic.

ATCL also provides commands that provide exposure to SkyWalker *Instrument Display* interface. These commands are implemented simply as additional ATCL commands that allow Client software to implement an *Instrument Display* on screen.

Astrometric Instruments, Inc. maintains ATCL documentation and Maestro-Lite source code and documentation at www.astrometric.com/support/resources/ATCL.html.

Finally, SkyWalker also supports a subset of the Meade LX200 variant of the ACL (Astronomical Control Language). Interface to SkyWalker, through its Client interface (i.e. Com port), can be via ATCL **or** ACL.

Document conventions

Astrometric-specific terms are *italicized* and definitions for each are provided in the "Glossary of terms" section at the end of this document.

Related documentation

This document refers to SkyWalker-specific features and terms. The "SkyWaker User's Manual" describes these features and defines the terms in detail. Therefore, it is strongly recommended that the "SkyWaker User's Manual" is read prior to this document.

SkyWalker's Client Interface

SkyWalker's Client interface provides three means of accessing SkyWalker through serial communications:

1. ATCL: the Astrometric Telescope Control Language: a general-purpose, open-standard means of controlling/polling SkyWalker.
2. The *Instrument Display* sub-set of ATCL commands provides a means for a small and simple "instrument display screen" with minimal buttons to interface to SkyWalker. Primarily this is used to create an HP2 emulator in Client software.
3. ACL: the Astronomical Control Language. The command syntax for ACL commands, supported by SkyWalker, is Meade LX200 compatible. SkyWalker supports a large subset of the LX200 protocol as described in the "SkyWalker User's Manual".

   The ACL protocol is **not** full-reply (ATCL **is** full-reply as described in the next section) and therefore Client software cannot intermingle ACL commands with ATCL commands. If SkyWalker detects ACL (via an ACL_ACK character), on its Client port, it will immediately enter ACL-mode. In ACL-mode, SkyWalker simply emulates a Meade LX200. To get SkyWalker out of ACL-mode, and into ATCL-mode, the Client must send a single ATCL_ENTER special character (defined below) and successfully receive back an ATCL_ACK special character from SkyWalker.

   Note: no further treatment of ACL is provided in this document. See the "SkyWalker's User's Manual" for full details.

1. Basic functional overview

1.1. Protocol Introduction

All ATCL commands have a return value indicating success, failure, or returning requested data. There are **no** commands that do not provide at least a simple success/fail return value. All Client applications should behave within the bounds of this "full-reply" protocol. This is to say that a Client should send a command and then **should not send another command until** it gets a reply from the first command. There are no reply-less ATCL commands. This is necessary for system integrity and Client management and tracking of which reply or return value is for which command. SkyWalker also requires the full-reply approach to assure that it can handle ATCL traffic under all conditions (i.e. SkyWalker can delay command processing if necessary to complete another higher priority task).

Note: if the Client sends another command before the former completes then SkyWalker sends an ATCL_CMND_OVERRUN message (more on messaging below).

Return values always take one of three forms (note: special characters presented here are defined in the following sections):

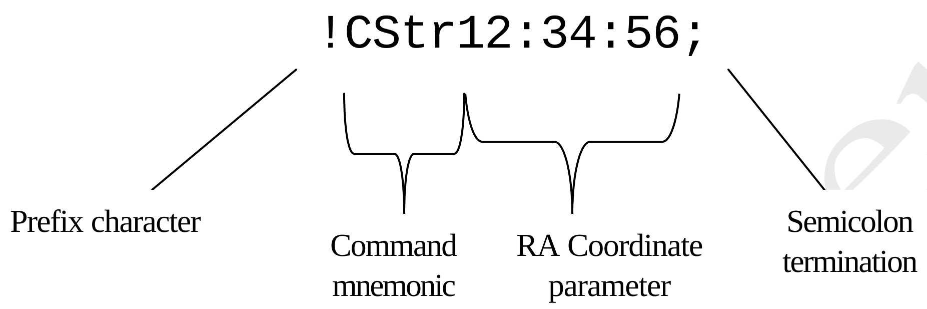
1. Reception of the command was successful however there was no return data (e.g. not a status command). In this case the single special character ATCL_ACK is returned. Note: there are additional single character return values for commands from the "Instrument Display" ATCL command subset as described in the section of that title below.
2. Reception of the command was successful and there was return data (e.g. a status command). In this case a string, terminated with the ';' (semicolon) character, is returned. The string contains the results of the status query. The string is always terminated with the semicolon character. ATCL syntax uses the semicolon character only as a string termination character.
3. Reception of the command was not successful either because of communications error or syntax error. In this case the single special character ATCL_NACK is returned.

In addition to replies and return-data from Client-initiated ATCL commands, ATCL also supports SkyWalker-initiated asynchronous messages that may be sent to the Client at any time. These messages include SkyWalker's Status, Warnings, Alerts, and Errors but can also include other data such as text update for the Client's *Instrument Display*, reporting of command syntax errors (in more detail than the ATCL_NACK return value provides), or communications errors.

1.2. ATCL Command Syntax introduced

ATCL command syntax follows the form "!nnnnpppp;", where the quotation marks are not part of the command, '!' is the prefix character, "nnnn" is the four character command mnemonic, "pppp" is the character parameter string (not all commands take parameters) and a ';' (i.e. semicolon) character **always** completes the command.

An example ATCL command is the **SetTargetRA** command which is used, prior to a GoTo, Alignment or Calibration, to set SkyWalker's internal Right Ascension target buffer. An example SetTargetRA command, in its entirety, appears as follows:



Multiple '!' will cause an ATCL_CMND_OVERRUN error. Multiple semicolon termination characters are ignored. Missing a semicolon termination character will eventually result in an ATCL_CMND_TIMEOUT error and SkyWalker will purge all characters received since the last '!' from its input buffer.

There is one, and only one, exception to the above command format. This is for the ATCL_ENTER command, which is a one-character command used to take SkyWalker out of ACL-mode (i.e. LX200 emulation mode). It is one-character so that SkyWalker can parse for it when in ACL-mode.

Return strings are straight ASCII printable characters. Semicolon-terminated so the Client knows when the return string ends. Once the semicolon character of a return string is received, the Client is free to send another command.

Detailed ATCL command syntax is provided in the "ATCL Syntax Reference" section below.

2. Advanced functionality

2.1. Asynchronous Messaging

As mentioned above, ATCL supports SkyWalker-initiated asynchronous messages that can be sent to the Client at any time. A unique non-printable ASCII character (i.e. ASCII character with bit7 set to '1') precedes all asynchronous messages. This is so that they can be "picked out" of the normal stream of replies coming back to the Client.

ATCL's messaging feature provides important information to the Client that is not available in the normal Client command return value. All messages that include a string (i.e. some messages are only single character) are semicolon-terminated and are preceded with a special character that allows the Client to distinguish the message from SkyWalker return data. Messages can occur at anytime, except **never** in the middle of a command reply. They can (and often do!) occur between when the Client sends a command to SkyWalker and when the Client gets a reply back. Because these messages can occur at anytime, and are not necessarily associated with any ATCL command, they are called "asynchronous" messages. The Client must have logic to sort the asynchronous messages from command replies.

ATCL messages never exceed a length of 88 characters (excluding the preceding special character and the terminating semicolon). The Client's message display area (if one is implemented) should therefore have the ability to log messages up to 88 characters long.

In addition to displaying the message to the user, it is recommended that the Client follow these guidelines:

- ♦ Issue a sound for each message to alert the user. An archive of standard sound wave files is provided at www.astrometric.com/reasources/sounds.zip for this purpose:
  - ATCL_STATUS: Status.wav
  - ATCL_WARNING: Warning.wav
  - ATCL_ALERT: Alert.wav
  - ATCL_INTERNAL_ERROR: Error.wav

  There are a few special status messages that the Client may associate the following special sounds:
  - Accel.wav:
    - "Status: GoTo commenced."
    - "Status: GoTo Anchor commenced."
  - Decel.wav:
    - "Status: GoTo completed."

    - "Status: GoTo Anchor completed."

- Provide a message list where the text of each message is buffered and provided for display to the user.
- Provide a pop-up message for Warnings, Alerts and Errors. No pop-up is suggested for Status.

2.2. Special characters

Special characters are used in the ATCL protocol as command return values indicating the success/failure of a command and as a preamble characters preceding every ATCL asynchronous message. Special characters are simply an ASCII character with bit7 set to 1. The Client can distinguish the start of an asynchronous message from ordinary replies and return-data by parsing for the requisite special character that precedes each asynchronous message from SkyWalker.

Complete list of special characters:

Value	Macro	Description
0xB1	ATCL_ENTER	1-character command that instructs SkyWalker to immediately exit ACL-mode and enter ATCL-mode. This should be the first command sent to SkyWalker when the Client starts up.
0x8F	ATCL_ACK	Acknowledge received from SkyWalker when a command was successful and no return reply string results from the command.
0xA5	ATCL_NACK(2)	Returned when the command was unsuccessful. An asynchronous message explaining the lake of success should occur.
0x9A	ATCL_STATUS	Character preceding a status asynchronous message
0x9B	ATCL_WARNING	Character preceding a warning asynchronous message
0x9C	ATCL_ALERT	Character preceding an alert asynchronous message

0x9D	ATCL_INTERNAL_ERROR	Character preceding an internal error asynchronous message (i.e. SkyWalker software or hardware failure). When an internal error occurs, SkyWalker is halted and a power-cycle is necessary. All subsequent ATCL commands to SkyWalker will result in a repeat of the internal error asynchronous message. The Client should cease sending commands following reception of an internal error and instruct the user to power-cycle SkyWalker.
0x9E	ATCL_SYNTAX_ERROR	There was a syntax error in an ATCL command. The offending command (lacking '!' prefix and ';' termination characters) is returned verbatim between the ATCL_SYNTAX_ERROR special character and semicolon termination character.
0x9F	ATCL_IDC_ASYNCH	Character preceding a "to Instrument Display" asynchronous packet. Described in the "Instrument Display asynchronous responses" subsection below.
0xA0	ATCL_VERIFY	Used when verification is needed from the Client. See ATCL Verify section below.
0xA1	ATCL_FRAMING(1)	There was a communications framing error.
0xA2	ATCL_COMM_OVERRUN(1)	There was a communications buffer overrun.
0xA3	ATCL_CMND_OVERRUN(1)	A new command was received prior to completion of the last command or prior to the full transmission of the last command's reply.
0xA4	ATCL_CMND_TIMEOUT(1)	A command was only partially formed within SkyWalker's command timeout period (1 second).
0xA6	ATCL_ID_CMND	See "Instrument Display interface" section below.
0xA7	ATCL_ID_LINK	See "Instrument Display interface" section below.
0xA8	ATCL_ID_DING	See "Instrument Display interface" section below.
0xAA	ATCL_CHANGE_NOTIFY	See Static Status Change Notification Section below.

0x06	ACL_ACK	A special character used in LX200 lingo. Use of this character is specifically avoided in the ATCL protocol and will immediately result in SkyWalker switching into ACL-mode. The Client must send a single-character ATCL_ENTER command to leave ACL-mode and re-enter ATCL-mode.

(1) These are special characters so that they are single character errors (i.e. they have no associated string, just a semicolon termination character) and hence have the best chance of being successfully communicated (from SkyWalker to the Client).

(2) ATCL_NACK essentially means that the command will not be executed as the Client intended. There are several possible reasons for this:

- Invalid command mnemonic
- Valid command mnemonic but parameter error (e.g. syntax error or out-off range, etc.)
- Valid command and valid parameter but illegal operation given present SkyWalker state (e.g. GoTo if not yet aligned)
- Valid command and valid parameter but the command was out of sequence (generally only possible with the firmware upgrade commands which must rigidly follow a specified sequence).

In all cases an asynchronous message will be issued that provides more information.

So, in short, ATCL_NACK means that SkyWalker could not act upon the command for one of the reasons listed above. Note, it is possible to send a command, for which SkyWalker replies with ATCL_ACK, that does not successfully complete the commanded action. The reason is that SkyWalker may execute the command, after checking it for validity, but encounter a reason later why the command can not be successfully completed. In this case an asynchronous message will be issued providing details on the operations failure.

2.3. Bandwidth and latency requirements

For certain ATCL Client applications (Maestro being an example), the refresh-rate of status on the Client's screen is difficult to support without special provisions in the ATCL syntax. For example, it is estimated that the Maestro Status Console Tab requires that approximately 25 ATCL status commands be issued on a regular interval (interval of ~1 second). Given the full-reply requirement of ATCL, this would imply that 25 ATCL-commands/reply-string pairs transpire in one second. While a very small bandwidth is necessary to accomplish this (est. ~400 characters/sec: easily accomplished with even a 9,600-baud connection), the sum of the PC's operating system latencies responding to these 25 transactions would likely vastly exceed one second (based on work with Windows Ring-3 response time done at Astrometric Instruments, Inc.).

Therefore, there should be no bandwidth limitations with ATCL running at the specified 19,200-baud rate however there could be severe latency issues in a Windows-based Client. To address the Windows latency issues, ATCL provides the "Static Status Change Notification" and "Status Aggregates" features as described in the next sections.

2.4. Static Status Change Notification

To reduce the communications bandwidth overhead of updating "static" status, and mitigate the need to continuously poll for updates, SkyWalker includes a "static status change notification" feature. The way this works, if enabled, is that whenever static status changes, an asynchronous reply indicating what has changed, and to what value, is sent to the Client. This feature is the complement to Dynamic Status Aggregates (see section of that name for details). Aggregates are a lower overhead way for Clients to poll SkyWalker's dynamic status. The static status change notification feature is a very low overhead way for Clients to receive notification of changed static status.

Static status is status that is not dynamic and does not change regularly. Static status only changes when a user makes a change. Since SkyWalker has multiple user interface inputs (e.g. handpaddle, Client port) a mechanism to notify the Client, when a static status change occurs, is a useful alternative to polling.

The static status change notification feature is enabled after the Client sends an EnableSSCN ("QEcn") command and disabled (be default) at SkyWalker power-up and after the DisableSSCN ("QDcn") command.

If enabled, static status change notification works as follows: when static status changes internal to SkyWalker an asynchronous message proceeded with the ATCL_CHANGE_NOTIFY special character is issued. Following the special character is the 4-charater ATCL command that would have been used to "get" the changed status. Following that is an equal sign ('=') and the string that SkyWalker would return if the 4-character command were issued directly.

For example, if the encoder counts per rev in X was changed then the following change notification would be sent:

```
*EGcx=45,000
```

Where '*' is the preceding ATCL_CHANGE_NOTIFY special character, EGcx is the GetEncoderCountsPerRevX command mnemonic and 45,000 is the new counts per revolution value.

The command reference section of this document notes all ATCL commands (that are used to get static status) for which this feature applies (see note #1: This is a "notifying" command for SkyWalker's Static Status Change Notification feature).

Notes & additional requirements:

- It is required that the Client still does one run through its polling loop, for all static status that it supports, when starting up. This is required, rather than the idea that all change notifications fire after the EnableSSCN is received, because the later would results in a flood of outgoing SkyWalker communications that would likely overflow Client input buffers and complicate SkyWalker's output communications mechanisms.
- It is suggested that the Client runs its static status polling loop slowly in the background to catch any notifications that might have been missed.

- It is suggested that Maestro issues the EnableSSCN command whenever it "reconnects" with SkyWalker after the connection with SkyWalker has been registered as lost. This will prevent the problem where SkyWalker, if powered-down, would disable static status change notification without Maestro knowing it.

2.5. Status Aggregates

<Need description here>

2.6. ATCL Verify

Occasionally it is necessary to provide a verification dialogue between the user and SkyWalker. For example, it may be necessary for SkyWalker to ask the user how to proceed in certain instances. To accommodate this dialogue ATCL provides a verify feature. With this feature SkyWalker will send an ATCL_VERIFY special character to the client followed by a question/verify string. When the client receives the ATL_VERIFY special character it should present the string to the user in a dialog box with an OK and Cancel button. If the user presses OK then send the ATCL_VerifyOK ("QAvo") command to SkyWalker, otherwise send the ATCL_VerifyCancel ("QAvc") command to SkyWalker.

Note: ATCL_VERIFY is a convenience however the Client should continuously poll to see if there is a verify active incase the ATCL_VERIFY special character is missed. Two commands provide for this:

- GetATCL_VerifyActive ("QGva"): Returns "Yes" or "No"
- GetATCL_VerifyString ("QGvs"): Returns the last verification string sent to the Client (which was proceeded by the ATCL_VERIFY special character)

2.7. Error protection

Overrun and Framing error checking should be enabled at the Client's UART. SkyWalker performs both of these error checks and issues ATCL_FRAMING and ATCL_COMM_OVERRUN as necessary.

SkyWalker provides syntax error checking on all ATCL commands and issues an ATCL_SYNTAX_ERROR if there was an error.

SkyWalker provides "timeout" checks to guard against partially formed commands and to halt SkyWalker's sending of Asynchronous messages to an inactive Client. Refer to the next section for complete details

2.8. Timeouts

SkyWalker's Client interface, when working in ATCL mode, provides three types of timeouts:

2.8.1. Command timeout

A command timeout occurs if more than 1 second transpires between the reception of a '!' prefix character and the ';' command completion character. The command timeout protects against mis-sequenced commands if communications with the Client software is lost (i.e. potentially leaving a partially-formed ATCL command). In the event of a command timeout, the ATCL_CMND_TIMEOUT single character Asynchronous Message is issued and SkyWalker's command input buffer is purged. Command timeouts are by default enabled at SkyWalker power-up however they can be disabled at anytime with the DisableCommandTimeOut command and re-enabled with the EnableCommandTimeOut command.

2.8.2. Client update timeout

It is desirable that SkyWalker issues no Asynchronous Messages if there is no Client connected to SkyWalker. The impetus for this behavior is two fold: 1) PC-based communications hardware can suffer buffer overflow, and subsequent dire consequences, if there is no Client software to service the serial port connected to SkyWalker (particularly a problem on some low-cost USB-to-RS232 converters) and 2) the communications DLL available as an interface between the Client and the SkyWalker demo program has a finite "depth" to its output buffer which will overflow with continuous update if there is no Client to "unload" it regularly.

To detect the presence or absence of a Client, SkyWalker includes a client update timeout. A client update timeout will occur if no ATCL command has been received within 5 seconds. A client update timeout is silent (i.e. there is no Asynchronous Message sent to the Client indicating that the timeout occurred) however, after a timeout occurs, SkyWalker will not issue the following Asynchronous Message to the Client until a ATCL command is received from the Client:

- ATCL_STATUS
- ATCL_WARNING
- ATCL_IDC_ASYNCH
- ATCL_CHANGE_NOTIFY

Note: SkyWalker is silent at power-up by default and will not issue the above messages. SkyWalker comes out of silence after reception of the first ATCL command from the Client.

2.9. Serial communications settings

As mentioned in the "Bandwidth and latency requirements" sub-section, the baud rate requirements for ATCL are low and the overall performance is likely to be more a function of OS communications latency. Therefore, a fixed low rate of 19,200-baud has been chosen for SkyWalker's Client RS232 interface (available from the connector labeled "Comm" on SkyWalker).

Note: we have considered the possibility of allowing higher baud rates in the future however SkyWalker will always power-up with its Client port configured for 19,200-baud. ATCL may add commands in the future that would provide for baud rate change.

ATCL requires that the Client be setup for 8-bit data, 1 stop bit with no parity checking enabled.

2.10. Client software design implications

With the ATCL protocol background information provided above, it is worthwhile summarizing the implications on Client software design.

2.10.1. Client commands must "take turns"

Firstly, the Client must not send new commands until it receives the full reply to a previous command. This simplifies Client design since there is a "lock step" correspondence between commands and replies.

2.10.2. Receive asynchronous messages

The Client must have the ability to receive messages, which are asynchronous to any command/reply pair. A special character, that makes their identification possible, always precedes messages.

2.10.3. Start by getting out of ACL mode and checking SkyWalker's firmware version

SkyWalker always powers-up in ACL (i.e. LX200-emulation) mode. This is necessary since Client software that is only LX200-compatible (and not SkyWalker-compatible) may not send the ACL_ACK character before any other commands. Therefore, the first ATCL command that the Client software sends to SkyWalker should be the ATCL_ENTER single-character command to be sure SkyWalker is ready to work in ATCL mode. In fact, the Client should continuously poll for SkyWalker's presence by sending the ATCL_ENTER single-character command until an ATCL_ACK command is received.

Note: this ATCL_ENTER polling should not be at too fast a rate to avoid SkyWalker input communications errors during power-up. A polling rate of once per second is nominal.

Next, the Client application should query SkyWalker's firmware version. This is an important next step since if SkyWalker is left inoperable (e.g. after a malfunctioned firmware upgrade) then it will reply with a version of "0.00.000" in response to the GetSkyWalkerFirmwareVersion ("HGfv") command. If SkyWalker reports a version of "0.00.000" then SkyWalker's firmware must be upgraded before it will function. The Client should either:

a) Proceed directly to upgrade SkyWalker's firmware to a valid version. For information on ATCL commands to upgrade SkyWalker's firmware please contact Astrometric Instruments, Inc.. Or…

b) If the Client does not support firmware upgrade, recommend that the user run Astrometric's SkyWalker Client **Maestro** or Astrometric's firmware upgrade utility **SW_Firmup** to upgrade the software.

Note: SkyWalker will also reply with a version of "0.00.000", in response to the GetSkyWalkerFirmwareVersion ("HGfv") command, following use of the FlashInvalidate ("QFiv") command.

2.10.4. Properly handle error conditions

Typical PC serial communications hardware has a limited hardware input buffer that stores-up incoming characters until the PC's operating system can find the time to un-load the buffer. If the PC's operating system is not fast enough in servicing the hardware input buffer then characters can be lost. Specifically, incoming characters, from SkyWalker, can cause a buffer overflow. Therefore, the Client must monitor the Overrun (and Framing) errors that the PC's UART provides. If an error is detected then the Client should flush the communications buffers, clear the error flags, and re-send the last un-replied command (if one exists) and carry on. This is a retry approach rather than a give-up approach and is preferable from the user's perspective. Error logging is recommended so that the user is aware of a bad interface.

Note: upon detection of a communications error the Client should also refresh the Instrument Display (the ID_Refresh command) since the errored and flushed packet may have been an Instrument Display update asynchronous response, which, if lost, would leave the Instrument Display in a different state then SkyWalker's internal representation of the Instrument Display.

Note: some CCD camera control software will disable the PC's interrupts! This particularly drastic action can result in communications errors, particularly Overrun errors since the PC's processor cannot always reliably service the serial communications hardware input buffer.

Note: if SkyWalker seems unresponsive for any reason then it may be because it has suffered an InternalError (i.e. SkyWalker software or hardware failure). If this is the case then SkyWalker will respond with an internal error asynchronous message (i.e. SkyWalker software or hardware failure). All subsequent ATCL commands to SkyWalker will result in a repeat of the internal error asynchronous message. The Client should cease sending commands following reception of an internal error and instruct the user to power-cycle SkyWalker (i.e. a power-cycle is necessary to resume normal operation).

2.10.5. Managing SkyWalker's presence

The serial communications link between SkyWalker and the Client can be lost at anytime. This can result in a partially formed reply to a command or asynchronous message (i.e. the semicolon character has not completed the SkyWalker-to-Client return packet yet). The Client should implement two timeouts to guard against a "hang" in either of these events:

1. A command reply timeout checks that a command is replied to. A reasonable limit of say 1 second for a completed command reply is suggested. Any longer than that and there is likely something wrong with the communications with SkyWalker.
2. An asynchronous message completion timeout would signal that the start of an asynchronous message was received however the semicolon termination character was not received in a reasonable time (e.g. 1 second).

Upon detection of a timeout it is suggested that the Client go into a "looking for SkyWalker" mode where a benign command is regularly send to SkyWalker (e.g. CommEcho or GetSkyWalkerSoftwareVersion). If an error-free response is received then the Client can leave the "looking for SkyWalker" mode and carry on.

Note: these measures are particularly important in the no-guaranteed-latency environment of the Microsoft Windows operating system.

2.10.6. Implement an Instrument Display interface

As described in the "Instrument Display interface" section below, ATCL includes a sub-set of functionality that exposes SkyWalker *Instrument Display* interface. It is recommended that any SkyWalker Client application, that is designed to provide full exposure to SkyWalker's features, expose the *Instrument Display* interface and properly deal with the extra single-character return codes that the *Instrument Display* provides.

Astrometric Instrument's open source Maestro-Lite program is an example of a Client that provides an *Instrument Display* interface.

2.10.7. Characters to disallow in strings from the user

The characters '!', ';' and '*' have important syntactical use in ATCL and/or SkyWalker internals as prefix and command/string termination characters. Client software should assure that no extraneous '!', ';' or '*' characters are present in command strings. It is of particular importance to filter-out '!' characters otherwise a user could inadvertently enter an '!' character in, for example, a site name. If the Client software did not remove it then the command exchange between SkyWalker and the Client would be mis-sequenced.

2.10.8. Be aware of ATCL's timeouts

The Client must send a complete command within the 1 second command timeout (i.e. from reception of '!' to reception of ';' must be less than 1 second) otherwise the command will be lost and an ATCL_CMND_TIMEOUT error will occur. To disable this timeout use the DisableCommandTimeOut command.

The Client must send a command to SkyWalker at least every 5 seconds otherwise SkyWalker will disable its Asynchronous Message reporting (i.e. "updating") to the Client. Reporting is re-enabled upon reception of any command. To disable this timeout use the DisableUpdateTimeOut command.

As mentioned in the "Managing SkyWalker's presence" sub-section above, the Client should implement its own timeouts to detect communications problems with SkyWalker. This is particularly important to prevent the Client from "locking up" in the event that communications with SkyWalker is lost while the Client awaits a reply.

3. ATCL Command Reference

The syntax of every ATCL command is simply "!gcccppp;". The '!' is the required prefix character, "gccc" is the command mnemonic (which is always 4 characters) where "g" is the command group character and "ccc" is unique within the command group, 'ppp' is 0 or more command parameters. All command must be terminated with a ';' (i.e. semicolon) character. **IT IS VERY IMPORTANT** to remember to apply a semicolon character to the end of the command otherwise an ATCL_CMND_TIMEOUT error will result.

For every ATCL command, SkyWalker returns something, either actual data (in the case of the Status commands for example) or a special character indicating success or lack of success. If the command was successful, and return data is not involved, then the single special character ATCL_ACK is returned (for *Instrument Display* commands, there are further special character success codes). If the command was not successful, then the single special character ATCL_ACK is returned. Any errors resulting from the command (e.g. syntax errors, system alerts, communications errors, etc.) are signaled through ATCL's asynchronous messaging scheme described in the sub-section "Asynchronous Messaging" above.

3.1. Command group list

Command group	Command group character
Alignment and Calibration	A
Coordinate query	C
(Axial) Encoder specific	E
General preferences	F
GoTo settings, initiation and status	G
Hardware control and status	H
Instrument Display	I
Handpaddle and autoguider	K
Time tools	L
Motors and gearing	M
Mount settings	N
Object database	O
Pointing model	P
Administrative and overhead	Q
Tracking settings	R
Sites settings	S
Time and date	T
User lists	U
Mark and return	V
Motion control and status	X
Reserved for future use	B, D, J, W, Y, Z

3.2. Command notes and conventions

3.2.1. Notes

- ♦ There are two basic types of ATCL commands: Set and Get. Set commands take one (and only one) parameter and Get commands return one (and only one) value.
- ♦ ATCL syntax is designed so that command parameters are as "atomic" as possible. That is to say that the parameters provide single values rather than multiple values. For example, commands such as

  SetTargetRA <RA>, SetTargetDec <Dec> and GoToTargetRA_Dec exist rather than

  GoToRA_Dec <RA> <Dec>

This is so that SkyWalker can test syntax at a more atomic level.

- ♦ Command mnemonics ARE case sensitive.
- ♦ Command parameters are NOT case sensitive.

3.2.2. Conventions

Conventions for formats specified in the "Parameters or return" column are interpreted as:

- ♦ SIGNED_2DIGIT:
  - For CoordFormat = Precise: sDD:MM:SS within range -90:00:00 to 90:00:00
  - For CoordFormat = Standard: sDD:MM within range -90:00 to 90:00
- ♦ SIGNED_3DIGIT:
  - For CoordFormat = Precise: sDDD:MM:SS within range -180:00:00 to +180:00:00
  - For CoordFormat = Standard: sDDD:MM within range -180:00 to +180:00
- ♦ UNSIGNED_3DIGIT:
  - For CoordFormat = Precise: DDD:MM:SS within range 000:00:00 to 360:00:00
  - For CoordFormat = Standard: DDD:MM within range 000:00 to 360:00
- ♦ HOURS
  - For CoordFormat = Precise: HH:MM:SS within range 00:00:00.0 to 24:00:00.0
  - For CoordFormat = Standard: HH:MM within range 00:00 to 24:00

- LATITUDE: in the form DD:MM:SSh, where 'h' is hemisphere (i.e. 'N' or 'S'). Within the range 90:00:00N to 90:00:00S
- LONGITUDE: in the form DDD:MM:SSh, where 'h' is hemisphere (i.e. 'E' or 'W'). Within the range 180:00:00W to 180:00:00E
- TIMEZONE: HH:MMh, where 'h' is hemisphere (i.e. 'E' or 'W'). Within the range 13:00W to 13:00W. Note: there are some South Pacific islands near the international dateline that actually use 13:00E (e.g., Nuku'alofa, Tonga). Additionally: "0:00" is acceptable for GMT (i.e. no hemisphere designation necessary).
- SHORT_TIME: HH:MM:SS
- LONG_TIME: DDD,HH:MM:SS (where DDD is day number)
- sD.DD: decimal number with indicated number of integer and fractional digits. The range of acceptable values is detailed in command descriptions. "s" is sign and, if present, is optional if the associated value is positive.
- sNN,NNN: Integer number with comma (',') 10^3 delineation. The range of acceptable values is detailed in command descriptions. "s" is sign and, if present, is optional if the associated value is positive
- VIEW_VEL: View Velocity in the format XX.XX<units> where <units> can be "deg/sec", "amin/sec", "asec/sec" or "X".
- VIEW_ACCEL: View Acceleration in the format XX.XX<units> where <units> can be "deg/sec^2", "amin/sec^2" or "asec/sec^2" where the subscript '2' can be represented by the hex character 0x1E or 0xB2.
- SECONDS: XXX.XXXsec. Note: for purposes of timing events (such as duration of High Drive assertion) SkyWalker's internal timing resolution is 0.075sec therefore the SECONDS value is rounded to the nearest 0.075sec increment by SkyWalker for these uses.
- DEG: XX.Xxdeg – represents an angle
- DEG_PER_SEC: XX.XXXXdeg/sec.
- TEMP: temperature in the form XX.XdegC or XX.XdegF
- FLAG: "Yes" or "No"
- SIGN: "Positive" of "Negative"
- SIDE: "East", "West" or "Unspecified"

3.2.3. Additional conventions

- Leading zeros in a parameter (e.g. "072:34:44W" for longitude) are allowed but unnecessary. Leading zeros are also possible in a return coordinate from SkyWalker.
- Positive signed values (e.g. "12degC") can be sent either with or without a preceding '+' character.

- A space character can be substituted for the colon character (':') in all coordinate formats.
- A comma (',') character can be substituted for a decimal ('.') character (European convention).

3.3. Alignment and Calibration commands

Command name	**Code**	**Parameter or return**	**Description**
Alignment status commands			
GetAlignmentState ?	**AGas**	<none>	Returns “NotAligned”, “Preliminary” or “Complete”
GetTimeSinceAlignmentComplete	**AGta**	LONG_TIME	
? Indicates command supported through SkyWalker's Static Status Change Notification (SSCN) feature			

Command name	**Code**	**Parameter or return**	**Description**
Alignment settings			
CheckSiteTimeDateSetOnce	**ACst**	FLAG	Checks if SkyWalker's site settings, time and date have been set at least once. This is useful, during the alignment sequence, to screen for the common error of the user not setting site, time and date prior to first alignment. Returns “Yes” if site, time and date have each been set at least once and “No” otherwise. This command also prompts an ATCL_WARNING detailing which settings have not been made.
SetAlignmentSide	**ASas**	“East” or “West”	Specifies, for Meridian Avoidance of FULL, which side of the sky the subsequent alignment will be completed on.
GetAlignmentSide ?	**AGai**	SIDE	
SetAlignmentStartingPositionFull	**ASpf**	-->	See SkyWalker User's Manual for a list of Starting Position names. Note: the list of available names depends on Meridian Avoidance (i.e. Full, Lower or None). If using Full Meridian Avoidance, use this command.
GetAlignmentStartingPositionFull ?	**AGpf**	-->	See SkyWalker User's Manual for a list of Starting Position names.

SetAlignmentStartingPositionNonFull	**ASpn**	-->	See SkyWalker User's Manual for a list of Starting Position names. Note: the list available names depends on Meridian Avoidance (i.e. Full, Lower or None). If not using Full Meridian Avoidance, use this command.
GetAlignmentStartingPositionNonFull ?	**AGpn**	-->	See SkyWalker User's Manual for a list of Starting Position names.
AlignFromStartingPosition	**AFsp**	<none>	Performs a Preliminary alignment from the currently defined starting position.
AlignFromTargetRA_Dec	**AFrd**	<none>	Marks an alignment sighting from pre-set target (using SetTarget* commands) RA/Dec. If Meridian Avoidance is Full and Alignment Side has not been chosen (with SetAlignmentSide command) then a warning message will result and the alignment will fail. When using this command, it IS necessary to use the SetAlignmentSide command first. Note: target RA/Dec are assumed to be valid in the EpochOfEntry (use SetEpochOfEntry command).
AlignFromTargetRA_DecEpochNow	**AFrn**	<none>	Same as AlignFromTargetRA_Dec except that target RA/Dec are assumed to be valid for an Epoch of Now.
AlignFromTargetRA_DecCalcSide	**AFcs**	<none>	Marks an alignment sighting from pre-set target RA/Dec. If Meridian Avoidance is Full then the proper Alignment Side is calculated by SkyWalker (i.e. it is not necessary to use the SetAlignmentSide command). If the target RA/Dec fall outside of the Meridian limits, or in an “overlap zone” accessible from either Alignment Side, then a warning message will result and the alignment will fail. Note: target RA/Dec are assumed to be valid in the EpochOfEntry (use SetEpochOfEntry command).
AlignFromTargetRA_DecCalcSideEpochNow	**AFcs**	<none>	Same as AlignFromTargetRA_DecCalcSide except that target RA/Dec are assumed to be valid for an Epoch of Now.
AlignFromTargetAltAz	**AFlz**	<none>	Marks an alignment sighting from pre-set target (using SetTarget* commands) Alt/Az. If Meridian Avoidance is Full and Alignment Side has not been chosen (with SetAlignmentSide command) then a warning message will result and the alignment will fail. When using this command, it IS necessary to use the SetAlignmentSide command first.
AlignFromSelectedObject	**AFso**	<none>	Marks an alignment sighting using the coordinates of the selected object. Other details are identical as for the AlignFromTargetRA_Dec command.
AlignFromSelectedObjectCalcSide	**AFss**	<none>	Marks an alignment sighting using the coordinates of the selected object. Other details are identical as for the AlignFromTargetRA_DecCalcSide command.
AlignFromLastPosition	**AFlp**	<none>	Aligns from the Axial Coordinates (which persist from the last alignment, even if SkyWalker was powered-down). Note: a successful previous alignment is necessary even though SkyWalker might have been powered-down in the mean time.
AlignVoid	**AVoi**	<none>	Voids alignment (returns alignment state to “NotAligned”).

? Indicates command supported through SkyWalker's Static Status Change Notification (SSCN) feature

Command name	**Code**	**Parameter or return**	**Description**
Calibration			
CalFromTargetAltAz	**ACIz**	<none>	If refraction correction is enabled then the target coordinates are adjusted for refraction (and are thus treated as apparent coordinates). If refraction correction is disabled then the target coordinates are not adjusted for refraction (and are thus treaded as observed coordinates).
CalFromTargetRA_Dec	**ACrd**	<none>	The Epoch assumed for target RA/Dec is set with the SetEpochOfEntry command. If refraction correction is enabled then the target coordinates are adjusted for refraction.
CalFromTargetRA_DecEpochNow	**ACrn**	<none>	The Epoch assumed for target RA/Dec is EpochNow. If refraction correction is enabled then the target coordinates are adjusted for refraction.
CalFromLastGoTo	**AClg**	<none>	
CalFromEncoders	**ACfe**	<none>	
CalFromSelectedObject	**ACfo**	<none>	

Command name	**Code**	**Parameter or return**	**Description**
Park position			
MarkParkPosition	**AMpp**	<none>	Marks the current Axial coordinates as the Park position. Park position is stored in SkyWalker's nonvolatile memory.
GetAtPark ?	**AGak**	<none>	If the telescope has not moved since successful completion of the last GoToPark command then returns "Yes", otherwise returns "No".
SetParkMarkStatus	**ASpm**	FLAG	Not recommended to use: use MarkParkPosition
GetParkMarkStatus ?	**AGpm**	FLAG	
SetParkPositionAxialX	**ASpx**	SIGNED_3DIGIT	Not recommended to use… use MarkParkPosition
GetParkPositionAxialX ?	**AGpx**	SIGNED_3DIGIT	
SetParkPositionAxialY	**ASpy**	SIGNED_3DIGIT	Not recommended to use… use MarkParkPosition
GetParkPositionAxialY ?	**AGpy**	SIGNED_3DIGIT	
? Indicates command supported through SkyWalker's Static Status Change Notification (SSCN) feature			

Command name	**Code**	**Parameter or return**	**Description**
		Home Index setup	
SetHomeIndexX_DiscoverDir	**ASdx**	SIGN	Positive motion in X is towards the left when looking away from the Scope Pole. HomeDiscovered is cleared if this value is changed.
GetHomeIndexX_DiscoverDir ?	**AGdx**	SIGN	
SetHomeIndexY_DiscoverDir	**ASdy**	SIGN	Positive motion in Y is towards the Scope Pole. HomeDiscovered is cleared if this value is changed.
GetHomeIndexY_DiscoverDir ?	**AGdy**	SIGN	
HomeDiscover	**AHds**	<none>	Initiates a sequence to "discover" the Home Index in each telescope axis. When a Home Index is found its exact Axial coordinates are saved and available later for the HomeSeek command. This command is not available unless SkyWalker is aligned. The direction that SkyWalker moves to "discover" the Home Index is specified by the SetHomeIndexX_DiscoverDir and SetHomeIndexY_DiscoverDir commands. When HomeDiscover is run the telescope needs to be far enough away from the Home Index switches that the telescope has fully accelerated to 1.0deg/sec (or VelMax, whichever is slower) in both axes. HomeDiscovered and HomeSeekError are cleared if this command successfully starts the "discovery" operation. HomeDiscovered is set if this command successfully completes the "discovery" operation and HomeSeekError is set if this command is unsuccessful. Note: this command requires that Automation Modules are in the system.
SetHomeDiscovered	**AShd**	FLAG	Sets HomeDiscovered state to FLAG value. Not recommended to use... use HomeDiscover
GetHomeDiscovered ?	**AGhd**	FLAG	
SetHomeIndexAxialX	**AShx**	SIGNED_3DIGIT	Sets AxialX coordinate of Home Index. Not recommended to use... use HomeDiscover
GetHomeIndexAxialX ?	**AGhx**	SIGNED_3DIGIT	
SetHomeIndexAxialY	**AShy**	SIGNED_2DIGIT	Sets AxialY coordinate of Home Index. Not recommended to use... use HomeDiscover
GetHomeIndexAxialY ?	**AGhy**	SIGNED_2DIGIT	
GetHomeSeekError ?	**AGhe**	FLAG	See HomeDiscover and HomeSeek commands.

? Indicates command supported through SkyWalker's Static Status Change Notification (SSCN) feature

Command name	**Code**	**Parameter or return**	**Description**

Align from Home Index			
HomeSeek	**AHsk**	none	Initiates a sequence to find the Home Index in each telescope axis. When the Home Indexes are found in both axes the AtHome flag is set (accessible via the GetAtHome command), the Track Rate is set to Drift (to preserve an eventual AtHome condition) and any alignment is voided. The direction that SkyWalker moves to find the Home Index is specified by the SetHomeIndexX_DiscoverDir and SetHomeIndexY_DiscoverDir commands (and need to be consistent with the directions used for HomeDiscover). AtHome and HomeSeekError are cleared if this command successfully starts the "seek" operation. AtHome is set, and SkyWalker sets its Axial coordinates to the index coordinates discovered and saved with the HomeDiscover command, if this command successfully completes the "seek" operation. The Client can then submit an AlignFromHome command to complete the Align from Home Index Seek operation. HomeSeekError is set if this command is unsuccessful. Note: this command requires that Automation Modules are in the system.
GetAtHome ?	**AGah**	none	If the telescope has not moved since successful completion of the last HomeSeek command then returns "Yes", otherwise returns "No".
AlignFromHome	**AFhi**	none	Same as AlignFromLast but requires that AtHome is asserted. After a HomeSeek is completed it is recommended that this command be used to align the telescope.
? Indicates command supported through SkyWalker's Static Status Change Notification (SSCN) feature			

3.4. Coordinate commands

Command name	**Code**	**Parameter or return**	**Description**
Coordinate status			
GetRA	**CGra**	HOURS	Returns HOURS (or "N/A" if not yet aligned)
GetDec	**CGde**	SIGNED_2DIGIT	Returns SIGNED_2DIGIT (or "N/A" if not yet aligned)
GetHourAngle	**CGha**	HOURS	Returns HOURS (or "N/A" if not yet aligned)
GetAz	**CGaz**	UNSIGNED_3DIGIT	Returns UNSIGNED_3DIGIT (or "N/A" if not yet aligned)
GetAlt	**CGal**	SIGNED_2DIGIT	Returns SIGNED_2DIGIT (or "N/A" if not yet aligned)
GetAxialRealX	**CGax**	SIGNED_3DIGIT	Returns SIGNED_3DIGIT_REAL
GetAxialRealY	**CGay**	SIGNED_3DIGIT	Returns SIGNED_3DIGIT_REAL
GetAxialVelocityX	**CGvx**	DEG_PER_SEC	Returns DEG_PER_SEC

GetAxialVelocityY	**CGvy**	DEG_PER_SEC	Returns DEG_PER_SEC
GetTargetAxialX	**CGtx**	SIGNED_3DIGIT	Returns SIGNED_3DIGIT
GetTargetAxialY	**CGty**	SIGNED_2DIGIT	Returns SIGNED_2DIGIT
GetScopeOrientation	**CGso**	-->	Returns “Primary” or “Secondary” (or “N/A” if not yet aligned)
SetCoordFormat	**CScf**	-->	Determines format that coordinates are presented in (see “Formats” section). Choices are “Standard” or “Precise”.
GetCoordFormat ?	**CGcf**	-->	Returns “Standard” or “Precise”.
GetAutomaticMoveStatus ?	**CGam**	FLAG	Returns “Yes” if the telescope is (automatically) seeking a target (e.g. GoTo, Home Index, Park), otherwise “No” is returned. Note: after SkyWalker process a targeting command it can take up to 1 second before GetAutomaticMoveStatus returns “Yes”. Note: even for a failed targeting command this command will issue an SSCN. This is so that the client can determine, from SSCN, if the targeting command failed as well as succeeded.
GetCoordinateAggregate1	**CGa1**	-->	Returns present RA, Dec, Hour angle, Az, Alt, Airmass and Refraction correction magnitude in the format: “HOURS|SIGNED_2DIGIT|HOURS|UNSIGNED_3DIGIT|SIGNED_2DIGIT|DD.D|DD.DDamin”
GetCoordinateAggregate2	**CGa2**	-->	Returns AutomaticMoveStatus, ScopeOrientation, AxialRealX, AxialRealY, AxialVelocityX, AxialVelocityY in the format: “FLAG|oooooo|SIGNED_3DIGIT_REAL|SIGNED_3DIGIT_REAL|XX.XXdeg/sec|XX.XXdeg/sec” where “oooooo” is “Primary” or “Secondary”.
? Indicates command supported through SkyWalker's Static Status Change Notification (SSCN) feature			

Command name	**Code**	**Parameter or return**	**Description**
			Target coordinates
SetTargetDec	**CStd**	SIGNED_2DIGIT	Note: be certain that the correct EpochOfEntry has been set with the SetEpochOfEntry command.
GetTargetDec	**CGtd**	SIGNED_2DIGIT	
SetTargetRA	**CStr**	HOURS	Note: be certain that the correct EpochOfEntry has been set with the SetEpochOfEntry command.
GetTargetRA	**CGtr**	HOURS	
SetTargetAlt	**CStl**	SIGNED_2DIGIT	Sets target Observed (not Apparent) Altitude.
GetTargetAlt	**CGtl**	SIGNED_2DIGIT	
SetTargetAz	**CStz**	UNSIGNED_3DIGIT	Sets target Observed (not Apparent) Azimuth.
GetTargetAz	**CGtz**	UNSIGNED_3DIGIT	

3.5. Encoder commands

Command name	**Code**	**Parameter or return**	**Description**
			Encoder settings
SetEncoderCountsPerRevX	**EScx**	NN,NNN	Acceptable range is 100 to 99,999,998
GetEncoderCountsPerRevX ?	**EGcx**	NN,NNN	
SetEncoderCountsPerRevY	**EScy**	NN,NNN	Acceptable range is 100 to 99,999,998
GetEncoderCountsPerRevY ?	**EGcy**	NN,NNN	
SetEncoderEnabledX	**ESnx**	FLAG	If "Yes" then the X encoder is enabled for use with the CalFromEncoder feature (see CalFromEncoder command in "Alignment and Calibration commands" section) and the Axial Encoder Track Lock feature.
GetEncoderEnabledX ?	**EGnx**	FLAG	
SetEncoderEnabledY	**ESny**	FLAG	If "Yes" then the Y encoder is enabled for use with the CalFromEncoder feature (see CalFromEncoder command in "Alignment and Calibration commands" section) and the Axial Encoder Track Lock feature.
GetEncoderEnabledY ?	**EGny**	FLAG	
SetEncoderPolarityX	**ESpx**	SIGN	
GetEncoderPolarityX ?	**EGpx**	SIGN	
SetEncoderPolarityY	**ESpy**	SIGN	
GetEncoderPolarityY ?	**EGpy**	SIGN	
? Indicates command supported through SkyWalker's Static Status Change Notification (SSCN) feature			

Command name	**Code**	**Parameter or return**	**Description**
			Target coordinates
SetAxialEncoderTrackLockEnabled	**EStl**	FLAG	When AETL is enabled, the error desired_position - encoder_position is continuously calculated. When this difference exceeds +/-1 encoder count then +/- motor steps are added to "catch up" the telescope axes (as measured by encoder_position) with desired_position.
GetAxialEncoderTrackLockEnabled ?	**EGtl**	FLAG	

SetAxialEncoderTrackLockCorrX	**ESax**	NN,NNN	Number of motor steps to correct X motor position when a discrepancy with encoder position is detected. Default value is 1 and max is 10. Note: this value can be set to 0 and then GetAxialEncoderTrackLockErrorX used to view un-corrected motor/encoder error.
GetAxialEncoderTrackLockCorrX ?	**EGax**	NN,NNN	
SetAxialEncoderTrackLockCorrY	**ESay**	NN,NNN	Number of motor steps to correct Y motor position when a discrepancy with encoder position is detected. Default value is 1 and max is 10. Note: this value can be set to 0 and then GetAxialEncoderTrackLockErrorY used to view un-corrected motor/encoder error.
GetAxialEncoderTrackLockCorrY ?	**EGay**	NN,NNN	
SetAxialEncoderTrackLockCadence	**ESac**	NN,NNN	Number of Ticks (75 milliseconds) between application of corrections
GetAxialEncoderTrackLockCadence ?	**EGac**	NN,NNN	Number of Ticks (75 milliseconds) between application of corrections
GetAxialEncoderTrackLockErrorX	**EGtx**	NN,NNN	The maximum number of encoder counts of absolute error between encoder position and motor position. This value is cleared a) each time this command is called and b) when Axial Encoder Track Lock is not enabled.
GetAxialEncoderTrackLockErrorY	**EGty**	NN,NNN	The maximum number of encoder counts of absolute error between encoder position and motor position. This value is cleared a) each time this command is called and b) when Axial Encoder Track Lock is not enabled.

? Indicates command supported through SkyWalker's Static Status Change Notification (SSCN) feature

Command name	**Code**	**Parameter or return**	**Description**
			Encoder diagnostics
GetEncDiagPositionX	**EGex**	NN,NNN	Returns an integer indicating the number of encoder counts detected since the last ZeroEncDiagPositionX command
ZeroEncDiagPositionX	**EZex**	<none>	Zeros EncDiagPositionX count
GetEncDiagPositionY	**EGey**	NN,NNN	Returns an integer indicating the number of encoder counts detected since the last ZeroEncDiagPositionY command
ZeroEncDiagPositionY	**EZey**	<none>	Zeros EncDiagPositionY count

? Indicates command supported through SkyWalker's Static Status Change Notification (SSCN) feature

3.6. Preference commands

Command name	**Code**	**Parameter or return**	**Description**

GoTo initiation			
SetUnitsFormat	**FSuf**	-->	Imperial or “Metric”
GetUnitsFormat ?	**FGuf**	-->	Returns “Imperial” or “Metric”
IncLED_Brightness	**FIlb**	<none>	Range is from 0 (i.e. off) to 8 (i.e. max). If at 8 then this command has no effect.
DecLED_Brightness	**FDlb**	<none>	Range is from 0 (i.e. off) to 8 (i.e. max). If at 0 then this command has no effect.
GetLED_Brightness ?	**FGlb**	N	Range is from 0 (i.e. off) to 8 (i.e. max).
? Indicates command supported through SkyWalker's Static Status Change Notification (SSCN) feature			

3.7. GoTo commands

Command name	**Code**	**Parameter or return**	**Description**
			GoTo initiation
GoToSelectedObject	**GOsl**	<none>	If refraction correction is enabled then the object coordinates are adjusted for refraction.
GoToTargetRA_Dec	**GTrd**	<none>	The Epoch assumed for target RA/Dec is set with the SetEpochOfEntry command. If refraction correction is enabled then the target coordinates are adjusted for refraction.
GoToTargetRA_DecEpochNow	**GTrn**	<none>	The Epoch assumed for target RA/Dec is EpochNow. If refraction correction is enabled then the target coordinates are adjusted for refraction.
GoToTargetAltAz	**GTlz**	<none>	Target coordinates are “observed” Alt/Az and are not adjusted for atmospheric refraction.
GoToLast	**GTol**	<none>	GoTo the coordinate of the “last” Goto
GoToPark	**GTop**	<none>	GoTo the preset Park position. Upon the completion of the GoTo the TrackRate is set to Drift and the AtPark flag is set. The status of alignment is unaffected.

Command name	**Code**	**Parameter or return**	**Description**
			GoTo restrictions
SetGoToHorizon	**GSgh**	DEG	Sets the minimum altitude allowed for GoTo targets. Allowable range is 0.0deg to “45.0deg”.
GetGoToHorizon ?	**GGgh**	DEG	

SetGoToOrientation	**GSgo**	-->	Primary, “Secondary”, “Current” or “Fastest”. Not available for Full Meridian avoidance or when the Up Soft Limit is used.
GetGoToOrientation ?	**GGgo**	-->	Primary, “Secondary”, “Current” or “Fastest”.
? Indicates command supported through SkyWalker's Static Status Change Notification (SSCN) feature			

Command name	**Code**	**Parameter or return**	**Description**
			GoTo status
GetGoToRemainPercent	**GGgr**	-->	100% to “0%”
GetGoToProgressPercent	**GGgp**	-->	0% to “100%”
GetNumberOfGoTos	**GGgn**	NN,NNN	Returns number string reporting number of GoTos since power-on

Command name	**Code**	**Parameter or return**	**Description**
			AutoCalibration
SetAutoCalEnabled	**GSac**	FLAG	Set SkyWalker's AutoCalibration feature enabled or disabled.
GetAutoCalEnabled ?	**GGac**	FLAG	
SetAutoCalBeyond	**GSab**	DEG	Sets the minimum distance to target which AutoCalibration will be used.
GetAutoCalBeyond ?	**GGab**	DEG	
? Indicates command supported through SkyWalker's Static Status Change Notification (SSCN) feature			

Command name	**Code**	**Parameter or return**	**Description**
			GoTo restrictions
SetConsistApproachGoToEnabled	**GSca**	FLAG	String “Yes” to enable and “No” to disable Consistent Approach GoTo
GetConsistApproachGoToEnabled#	**GGca**	FLAG	Returns “Yes” if Consistent Approach GoTo is enabled and “No” if disabled
SetConsistApproachGoToDirDec	**GScd**	-->	String “North” or “South” specifying the direction in which the approach is made for Polar mount types.
GetConsistApproachGoToDirDec#	**GGcd**	-->	String “North” or “South”

SetConsistApproachGoToOffset	**GSco**	-->	Arcminutes in the range 1-60 in the form “1.00amin”-”60.00amin”
GetConsistApproachGoToOffset#	**GGco**	-->	Returns arcminutes
? Indicates command supported through SkyWalker's Static Status Change Notification (SSCN) feature			

3.8. Hardware commands

Command name	**Code**	**Parameter or return**	**Description**
			TCS devices
GetSkyWalkerModel	**HGsm**	-->	Returns a string indicating the SkyWalker model.
GetSkyWalkerFirmwareVersion	**HGfv**	m.nn.rrr	Note: if “0.00.000” is returned then SkyWalker's firmware is unprogrammed and it is necessary to proceed directly to SkyWalker's firmware upgrade procedure since SkyWalker will not accept commands other than firmware upgrade commands.
GetSkyWalkerSerialNumber	**HGsn**	NN,NNN	
GetSkyWalkerFlashModel	**HGfm**	-->	Possible return strings: “IntelB3-4MB” or “IntelC3-4MB”
GetSkyWalkerSupplyVoltage	**HGvv**	-->	In the format: “D.Dvolts”
GetSkyWalkerSupplyVotlageStat	**HGvs**		Returns: - “OK” if supply voltage is above low-voltage warning point. - “Low” if supply voltage is between low-voltage warning point and under-voltage fault point. - “Under” if supply voltage is under the under-voltage fault point.
GetHP1_Present	**HG1p**	FLAG	
GetHP2_Present	**HG2p**	FLAG	
GetHREMX_Present	**HGhx**	FLAG	
GetHREMY_Present	**HGhy**	FLAG	
GetAutoModX_Present	**HGax**	FLAG	
GetAutoModY_Present	**HGay**	FLAG	
GetAutoModX_AuxIn	**HGix**	-->	Returns “Yes” if X (RA/Az) Automation Module AuxIn input is active (high) and “No” if inactive and “N/A” if Automation Module X is not present in the system.
GetAutoModY_AuxIn	**HGiy**	-->	Returns “Yes” if Y (Dec/Alt) Automation Module AuxIn input is active (high) and “No” if inactive and “N/A” if Automation Module Y is not present in the system.
SetAutoModX_AuxOutEnabled	**HSox**	FLAG	String “Yes” to enable and “No” to disable the Automation Module's AuxOut output

GetAutoModX_AuxOutEnabled#	**HGox**	-->	Yes, “No” or “N/A”, where “N/A” indicates that the Automation Module is not installed in the system
SetAutoModY_AuxOutEnabled	**HSoy**	FLAG	String “Yes” to enable and “No” to disable the Automation Module's AuxOut output
GetAutoModY_AuxOutEnabled#	**HGoy**	-->	Yes, “No” or “N/A”, where “N/A” indicates that the Automation Module is not installed in the system
GetSwitchProPresent	**HGwp**	FLAG	

? Indicates command supported through SkyWalker's Static Status Change Notification (SSCN) feature

Command name	**Code**	**Parameter or return**	**Description**
Fault status and control			
GetTopActiveFault	**HGtf**	-->	Returns a string representing the highest priority (i.e. top) active fault. “None” if no faults are presently active. Possible faults, and return strings (in priority order) are: - “HardLimitRA” - “HardLimitDec” - “UnderVoltage” - “StallRA” - “StallDec” - “DrvOverTempRA” - “DrvOverTempDec” - “MotOverTempRA” - “MotOverTempDec” - “OverCurrentRA” - “OverCurrentDec” - “OverSpeedRA” - “OverSpeedDec” - “None” Note: for Alt/Az mounts “RA” is replaced with “Az” and “Dec” is replaced with “Alt”
ClearEventFaults	**HCef**	<none>	Clears “event” faults. Event faults are associated with SkyWalker-Servo and are set in the hardware and require clearing from the user. Event faults include: - StallRA/Dec - OverCurrentRA/Dec - OverSpeedRA/Dec Note: for Alt/Az mounts “RA” is replaced with “Az” and “Dec” is replaced with “Alt”

HardLimitOverride	**HHlo**	<none>	Requires installation of Automation Module to operate. When this command is issued the Automation Module will override/deassert SkyWalker's HardLimit input for 5 seconds. With this command, a remote client can "back out" of a HardLimit condition in a controlled manner that does not require on-site operator intervention. For more details see the "Automation Module User's Manual" (DOC-14).
HardLimitBackOut	**HHlb**	<none>	Requires installation of Automation Module to operate. When this command is issued SkyWalker will attempt to move out of a HardLimit condition. The specific sequence that this command follows is: 1. Find first active Axial Limit in the sequence: East, West, Up, Down. If the positive and negative Axial Limits are both asserted for either X or Y then exit with "Both Axial Hard Limits in the X/Y axis engaged... cannot "back out".". 2. Signal the Automation Module to override SkyWalker's Hard Limit input for 5 seconds. 3. Issue an Axial GoTo with a destination that is 2 degrees inside the limit found in step 1. 4. After completion of GoTo, and 5 second Hard Limit override delay, if Hard Limit is no longer asserted then exit the sequence quietly, otherwise exit with "Axial Hard Limit still asserted after "back out" has completed.". Note: if no Hard Limits are active, or if SkyWalker is still in a previous Hard Limit "back out" sequence, this command does nothing.
? Indicates command supported through SkyWalker's Static Status Change Notification (SSCN) feature			

Command name	**Code**	**Parameter or return**	**Description**
			Axial (Hard) limits
GetAxialLimitEastActive	**HLse**	FLAG	Indicates if East Axial Limit is active
GetAxialLimitWestActive	**HLsw**	FLAG	Indicates if West Axial Limit is active
GetAxialLimitUpActive	**HLsu**	FLAG	Indicates if Up Axial Limit is active
GetAxialLimitDownActive	**HLsd**	FLAG	Indicates if Down Axial Limit is active
SetAxialLimitEastAngle	**HSae**	DEG	Angle measured to the East of the celestial meridian
GetAxialLimitEastAngle	**HGae**	DEG	
SetAxialLimitWestAngle	**HSaw**	DEG	Angle measured to the West of the celestial meridian
GetAxialLimitWestAngle	**HGaw**	DEG	
SetAxialLimitUpAngle	**HSau**	DEG	Angle measured Up from the Scope Equator
GetAxialLimitUpAngle	**HGau**	DEG	

SetAxialLimitDownAngle	**HSad**	DEG	Angle measured Down from the Scope Equator
GetAxialLimitDownAngle	**HGad**	DEG	
SetAxialLimitEastWestEnabled	**HSew**	FLAG	String “Yes” to enable and “No” to disable action when Axial Limits are struck for the X-axis.
GetAxialLimitEastWestEnabled ?	**HGew**	FLAG	
SetAxialLimitUpEnabled	**HSeu**	FLAG	String “Yes” to enable and “No” to disable action when the Up Axial Limit is struck.
GetAxialLimitUpEnabled ?	**HGeu**	FLAG	
SetAxialLimitDownEnabled	**HSed**	FLAG	String “Yes” to enable and “No” to disable action when the Down Axial Limit is struck.
GetAxialLimitDownEnabled ?	**HGed**	FLAG	
? Indicates command supported through SkyWalker's Static Status Change Notification (SSCN) feature			

Command name	**Code**	**Parameter or return**	**Description**
Dome			
SkyWalker's SwitchPro peripheral (or SkyWalker model supporting UniDrive outputs) or DomePro accessory is required for the following commands to function.			
SetDomePolarity	**HSdp**	SIGN	
GetDomePolarity ?	**HGdp**	SIGN	
DomeLeftOn	**HDlo**	<none>	Causes azimuth slewing motion “left” (i.e. decrease in azimuth)
DomeRightOn	**HDro**	<none>	Causes azimuth slewing motion “right” (i.e. increase in azimuth)
KillDomeAzimuthMovement	**HXdm**	<none>	Kill dome azimuth motion
Astrometric's DomePro accessory is required for the following commands to function. Note for DomePro commands which expect a numeric parameter require that number be in hexidecimal format. DomePro does not directly accept decimal values.			
SetDomeCPR	**HSdc**	HEX	Counts per revolution of the dome in azimuth. Hexadecimal value in the range “20” (32) to “40000000” (1,073,741,824) and must be an even number.
GetDomeCPR	**HGdc**	HEX	
SetDomeMaxVel	**HSde**	HEX	Bounded to a hexadecimal value of “1” to “7C” (“7C” hex is 124 decimal). A value of 124 decimal is equivalent to 124,000 steps per second. All other legal values scale linearly.
GetDomeMaxVel	**HGde**	HEX	

SetDomeAccel	**HSda**	HEX	Bounded to a hexadecimal value of "1" to "FF" ("FF" hex is 255 decimal). A value in the range of 1 to 255. A value of 255 decimal is equivalent to acceleration ramp to full speed (i.e. 124,000 steps/second) in ~1/8 seconds. All other legal values scale to a ramp time inverse linearly (smaller acceleration leads to larger ramp time in reverse linear manner).
GetDomeAccel	**HGda**	HEX	
GetDomePosition	**HGdo**	HEX	Azimuth step position
GetDomeAzMoveMode	**HGdz**	-->	Returns: - Fixed - Left -- moving left - Right -- moving right - GoTo - Homing
GetDomeShutterStatus	**HGds**	-->	Returns a character with the following possible values: - '0' for Opened - '1' for Closed - '2' for Opening - '3' for Closing - '4' for ShutterError - '5' if the shutter module is not communicating to the azimuth module
GetDomeLimits	**HGdl**	-->	Returns five characters: 12345, where each character can be '0' (inactive) or '1' (active): - First character (1): UpperShutterOpened switch state - Second character (2): UpperShutterClosed switch state - Third character (3): LowerShutterOpened switch state - Forth character (4): LowerShutterClosed switch state - Fifth character (5): AtHome bit. See DomeHome command.
DomeHome	**HDhm**	<none>	Moves to the Right until the Home Index switch is found.
DomeAzGoTo	**HDgo**	HEX	Accepts a signed 32-bit hexadecimal number
DomeOpenShutters	**HSso**	<none>	Start opening shutters if not already open
DomeCloseShutters	**HSsc**	<none>	Start closing shutters if not already closed
DomeKillShutterMovement	**HXsm**	<none>	Terminates all azimuth and shutter movement.
GetDomeProFirmwareVersion	**HGdf**	-->	Returns DomePro firmware version
GetDomeDebug	**HGdg**	-->	Returns an arbitrary string used for engineering/debug purposes
GetDomeSupplyVoltageAzimuth	**HGva**	HEX	Returns 8-bit hexadecimal voltage sense ADC value. Full range ("FF") value represents 15.0v. Any value below "1D" indicates that the shutdown input to the azimuth module is active.

GetDomeSupplyVoltageShutter	**HGvs**	HEX	Returns 8-bit hexadecimal voltage sense ADC value. Full range ("FF") value represents 15.0v. Any value below "1D" indicates that the shutdown input to the azimuth module is active.
? Indicates command supported through SkyWalker's Static Status Change Notification (SSCN) feature			

Command name	**Code**	**Parameter or return**	**Description**
Focus			
SkyWalker's SwitchPro peripheral (or SkyWalker model supporting UniDrive outputs) or FocsPro accessory is required for the following commands to function.			
SetFocusFastRate	**HSff**	NN	Rate is between SlowRate and 15
GetFocusFastRate ?	**HGff**	NN	
SetFocusSlowRate	**HSfs**	NN	Rate is between 1 and FastRate
GetFocusSlowRate ?	**HGfs**	NN	
SetFocusPolarity	**HSfp**	POLARITY	
GetFocusPolarity ?	**HGfp**	POLARITY	
SetFocusPulseTime	**HSfu**	SECONDS	0.075sec to 1.000sec
GetFocusPulseTime ?	**HGfu**	SECONDS	
ZeroFocusDisplacement	**HZfd**	<none>	
GetFocusDisplacement	**HGfd**	SECONDS	Amount of time that focus was On (slow or fast)
FocusFastOn	**HEfo**	-->	Out or "In"
FocusSlowOn	**HEso**	-->	Out or "In"
KillFocusMovement	**HXfc**		
FocusFastPulse	**HEfp**	-->	Out or "In" pulse, at focus FastRate, for duration set with SetFocusPulseTime command.
FocusSlowPulse	**HEsp**	-->	Out or "In" pulse, at focus SlowRate, for duration set with SetFocusPulseTime command.
Astrometric's FocusPro accessory is required for the following commands to function. Note for FocusPro commands which expect a numeric parameter require that number be in hexadecimal format. FocusPro does not directly accept decimal values.			
SetFocusMaxVel	**HSfe**	HEX	Bounded to a hexadecimal value of "1" to "80" ("80" hex is 128 decimal). A value of 128 decimal is equivalent to 32,000 microsteps per second. All other legal values scale linearly.
GetFocusMaxVel	**HGfe**	HEX	

SetFocusAccel	**HSfa**	HEX	Bounded to a hexadecimal value of "1" to "FF" ("FF" hex is 255 decimal). A value of 255 decimal is equivalent to acceleration ramp to full speed (i.e. 32,000 microsteps/second) in 0.512 seconds. All other legal values scale to a ramp time inverse linearly (smaller acceleration leads to larger ramp time in reverse linear manner).
GetFocusAccel	**HGfa**	HEX	
SetFocusCurrentReduction	**HSfn**	-->	Sets the current reduction when idle for more than 150 milliseconds. Choices are 1 (no reduction), 2 (1/2 current), 4 (1/4 current) or 0 (motor de-energized after 150 milliseconds). Note, if 0 is used then the motor rotor position will move to the nearest full-step and only motor detent torque will persist.
GetFocusCurrentReduction	**HGfn**		
GetFocusPosition	**HGfo**	HEX	Step position
GetFocusTemperature	**HGft**	HEX	Returns FocusPro's temperature Analog-to-Digital converter value in the range of 0-FF (0-255 digital). For the conversion table which converts this to sensor temperature please contact Astrometric Instruments.
GetFocusMoveMode	**HGfz**	-->	Returns: - Fixed - Fast - Slow - GoTo - Homing
GetFocusLimitsAndFaults	**HGfl**	-->	Returns four characters: 1234, where each character can be '0' (inactive) or '1' (active): - First character (1): HomeInLimit state - Second character (2): OutLimit state - Third character (3): Driver over-temperature state (note: if over-temperature is detected the driver will shutdown until power is cycled). - Forth character (4): AtHome bit. See FocusHome command.
FocusHome	**HFhm**	<none>	FocusPro initiates a four-step home-finding sequence: - Step1: move out fast until home switch not active - Step2: move in fast until home switch found - Step3: move out slow until fully accelerated and home switch not active - Step4: move in slow until home switch found. In step 4, when home is found all motion terminates and the AtHome bit is set (read with the GetFocusLimitsAndFaults command).
FocusGoTo	**HFgo**	HEX	Parameter is step position (>= '0" or <= '3FFFFF') to "go to".
GetFocusProFirmwareVersion	**HGff**	m.nn.rrr	Returns FocusPro firmware version
GetFocusDebug	**HGfg**	-->	Returns an arbitrary string used for engineering/debug purposes

? Indicates command supported through SkyWalker's Static Status Change Notification (SSCN) feature

Command name	**Code**	**Parameter or return**	**Description**
			Indicator
SetIndicatorEnabled	**HSie**	FLAG	
GetIndicatorEnabled ?	**HGie**	FLAG	
SetIndicatorBrightness	**HSib**	NN	Brightness is between 1 and 15
GetIndicatorBrightness ?	**HGib**	NN	
SetIndicatorOnTime	**HSio**	SECONDS	0.075sec to IndicatorPeriod
GetIndicatorOnTime ?	**HGio**	SECONDS	
SetIndicatorPeriod	**HSip**	SECONDS	IndicatorOnTime to 18.000sec
GetIndicatorPeriod ?	**HGip**	SECONDS	
? Indicates command supported through SkyWalker's Static Status Change Notification (SSCN) feature			

Command name	**Code**	**Parameter or return**	**Description**
			OneShot
SetOneShotOnTime	**HSoo**	SECONDS	0.075sec to 18.000sec
GetOneShotOnTime ?	**HGoo**	SECONDS	
OneShotFire	**HEos**	<none>	
GetOneShotStatus	**HGos**	-->	On or "Off"
? Indicates command supported through SkyWalker's Static Status Change Notification (SSCN) feature			

Command name	**Code**	**Parameter or return**	**Description**
			Dew Heater

SetDewHeater1_Enabled	**HSe1**	FLAG	String "Yes" to enable and "No" to disable
GetDewHeater1_Enabled ?	**HGe1**	FLAG	
SetDewHeater2_Enabled	**HSe2**	FLAG	
GetDewHeater2_Enabled ?	**HGe2**	FLAG	
SetDewHeater3_Enabled	**HSe3**	FLAG	
GetDewHeater3_Enabled ?	**HGe3**	FLAG	
SetDewHeater1_OnTime	**HSo1**	SECONDS	0.075sec to DewHeater1_Period
GetDewHeater1_OnTime ?	**HGo1**	SECONDS	
SetDewHeater2_OnTime	**HSo2**	SECONDS	0.075sec to DewHeater2_Period
GetDewHeater2_OnTime ?	**HGo2**	SECONDS	
SetDewHeater3_OnTime	**HSo3**	SECONDS	0.075sec to DewHeater3_Period
GetDewHeater3_OnTime ?	**HGo3**	SECONDS	
SetDewHeater1_Period	**HSp1**	SECONDS	DewHeater1_OnTime to "18.000sec"
GetDewHeater1_Period ?	**HGp1**	SECONDS	
SetDewHeater2_Period	**HSp2**	SECONDS	DewHeater2_OnTime to "18.000sec"
GetDewHeater2_Period ?	**HGp2**	SECONDS	
SetDewHeater3_Period	**HSp3**	SECONDS	DewHeater3_OnTime to "18.000sec"
GetDewHeater3_Period ?	**HGp3**	SECONDS	
? Indicates command supported through SkyWalker's Static Status Change Notification (SSCN) feature			

Command name	**Code**	**Parameter or return**	**Description**
Momentary			
SetMomentary1_Enabled	**HSm1**	FLAG	String "Yes" to set Momentary output and "No" to clear
SetMomentary2_Enabled	**HSm2**	FLAG	String "Yes" to set Momentary output and "No" to clear
SetMomentary3_Enabled	**HSm3**	FLAG	String "Yes" to set Momentary output and "No" to clear

Command name	Code	Parameter or return	Description
Toggle			
Toggle1	**HEt1**	<none>	Toggles the output
GetToggle1_Status ?	**HGt1**	FLAG	
Toggle2	**HEt2**	<none>	Toggles the output
GetToggle2_Status ?	**HGt2**	FLAG	
Toggle3	**HEt3**	<none>	Toggles the output
GetToggle3_Status ?	**HGt3**	FLAG	
? Indicates command supported through SkyWalker's Static Status Change Notification (SSCN) feature			

Command name	Code	Parameter or return	Description
Autoguider			
GetAutoGuiderState	**HGas**	-->	Returns a four character string where each character is '1' or '0' depending on weather the associated AutoGuider input is active or not. Encoding: Character # Direction indicated 1 Up 2 Down 3 Left 4 Right For example, "0101" means that Down and Right are active.

Command name	Code	Parameter or return	Description
HighDrive assignments			
SetBiDriveA_Assignment	**HSba**	<see below>	
GetBiDriveA_Assignment ?	**HGba**	<see below>	
SetBiDriveB_Assignment	**HSbb**	<see below>	
GetBiDriveB_Assignment ?	**HGbb**	<see below>	

SetUniDriveA_Assignment	**HSua**	<see below>	
GetUniDriveA_Assignment ?	**HGua**	<see below>	
SetUniDriveB_Assignment	**HSub**	<see below>	
GetUniDriveB_Assignment ?	**HGub**	<see below>	
SetUniDriveC_Assignment	**HSuc**	<see below>	
GetUniDriveC_Assignment ?	**HGuc**		
SetUniDriveD_Assignment	**HSud**	<see below>	
GetUniDriveD_Assignment ?	**HGud**	<see below>	
For the above commands, the following assignment parameters apply:	"Alarm", "AlwaysOff", "AlwaysOn", "DewHeater1", "DewHeater2", "DewHeater3", "DomeDir", "DomeDrive", "FocusDir", "FocusDrive", "Indicator", "MapLight", "Momentary1", "Momentary2", "Momentary3", "OneShot", "Timer", "Toggle1", "Toggle2", "Toggle3"		
? Indicates command supported through SkyWalker's Static Status Change Notification (SSCN) feature			

Command name	**Code**	**Parameter or return**	**Description**
HighDrive diagnostics			
SetDiagHighDriveEnabled	**HSdh**	FLAG	String "Yes" to enable and "No" to disable
GetDiagHighDriveEnabled ?	**HGdh**	FLAG	
DiagToggleBiDriveA	**HTba**	<none>	Does nothing if HighDrive diagnostics is disabled
DiagGetBiDriveA_State ?	**HDba**	FLAG	
DiagToggleBiDriveB	**HTbb**	<none>	Does nothing if HighDrive diagnostics is disabled
DiagGetBiDriveB_State ?	**HDbb**	FLAG	
DiagToggleUniDriveA	**HTua**	<none>	Does nothing if HighDrive diagnostics is disabled
DiagGetUniDriveA_State ?	**HDua**	FLAG	
DiagToggleUniDriveB	**HTub**	<none>	Does nothing if HighDrive diagnostics is disabled
DiagGetUniDriveB_State ?	**HDub**	FLAG	
DiagToggleUniDriveC	**HTuc**	<none>	Does nothing if HighDrive diagnostics is disabled
DiagGetUniDriveC_State ?	**HDuc**	FLAG	

DiagToggleUniDriveD	**HTud**	<none>	Does nothing if HighDrive diagnostics is disabled
DiagGetUniDriveD_State ?	**HDud**	FLAG	
? Indicates command supported through SkyWalker's Static Status Change Notification (SSCN) feature			

Command name	**Code**	**Parameter or return**	**Description**
Momentary			
GetInSystemAggregate1	**HGi1**	-->	Returns HP1_Present, HP2_Present, HREMX_Present, HREMY_Present, AutoModX_Present, AutoModY_Present in the format: "FLAG|FLAG|FLAG|FLAG|FLAG|FLAG"
GetInSystemAggregate2	**HGi2**	-->	Returns DomeProPresent, FocusProPresent, FilterProPresent, SwitchProPresent in the format: "FLAG|FLAG|FLAG|FLAG"
GetHardwareAggregate1	**HGh1**	-->	Returns SkyWalkerSupplyVoltage, SkyWalkerSupplyVotlageStatus, AutoGuiderState, TopActiveFault, TopLimitViolation, AutoModX_AuxIn and AutoModY_AuxIn in the format: "XX.Xv|sss|UDLR|ffffff|llllll|FLAG|FLAG"
GetHardwareAggregate2	**HGh2**	-->	Returns AxialLimitEastActive, AxialLimitWestActive, AxialLimitUpActive, AxialLimitDownActive in the format: "FLAG|FLAG|FLAG|FLAG"

3.9. Instrument Display commands

See the sub-section in section 4 titled “Instrument Display interface”

3.10. HandPaddle and AutoGuider commands

Command name	**Code**	**Parameter or return**	**Description**
Actions			
SetSlew	**KSsl**	<none>	Puts handpaddle in Slew Mode (takes it out of View Mode)
ClearSlew	**KCsl**	<none>	Removes handpaddle from View Mode (takes it out of Slew Mode)
GetSlew#	**KGsl**	FLAG	
StartPersistentUp	**KSpu**	-->	Accepts a string with "0" to "100" representing 0%-100% of the current max velocity (i.e. VelMax for Slew and current ViewVel for View). A string of "101" signals SkyWalker to hold the current velocity if accelerating.

StartPersistentDown	**KSpd**	-->	See StartPersistentUp
StartPersistentLeft	**KSpl**	-->	See StartPersistentUp
StartPersistentRight	**KSpr**	-->	See StartPersistentUp
ExeGoToAnchor	**KEan**		

Command name	**Code**	**Parameter or return**	**Description**
			View motion settings
SetCurrentViewVel	**KScv**	-->	Character '1', '2', '3' or '4'
GetCurrentViewVel ?	**KGcv**		
SetViewInvertEnabled	**KSvi**	FLAG	Invert flips Left and Right handpaddle buttons in View Mode
GetViewInvertEnabled ?	**KGvi**	FLAG	Returns "Yes" if enabled and "No" if disabled
SetViewMirrorEnabled	**KSvr**	FLAG	String "Yes" to enable and "No" to disable
GetViewMirrorEnabled ?	**KGvr**	FLAG	Returns "Yes" if enabled and "No" if disabled
SetViewAccel	**KSva**	VIEW_ACCEL	Min of 0.5000deg/sec2 max of AccelMaxX or AccelMaxY whichever is lower.
GetViewAccel ?	**KGva**	VIEW_ACCEL	
SetViewMode	**KSvm**	-->	PanScope or "PanRA/Dec" (note: only "PanScope" is supported in present SkyWalker firmware)
GetViewMode ?	**KGvm**	-->	See SetViewMode
SetViewVel1	**KSv1**	VIEW_VEL	Min of 1.0000asec/sec max of ViewVel2
GetViewVel1 ?	**KGv1**	VIEW_VEL	
SetViewVel2	**KSv2**	VIEW_VEL	Min of ViewVel1 max of ViewVel3
GetViewVel2 ?	**KGv2**	VIEW_VEL	
SetViewVel3	**KSv3**	VIEW_VEL	Min of ViewVel2 max of ViewVel4
GetViewVel3 ?	**KGv3**	VIEW_VEL	
SetViewVel4	**KSv4**	VIEW_VEL	Min of ViewVel3 max of 2.0000deg/sec or VelMaxX or VelMaxY whichever is lower.
GetViewVel4 ?	**KGv4**	VIEW_VEL	
SetAutoGuiderVel	**KSvg**	VIEW_VEL	Min of 1.0000asec/sec max of 2.0000deg/sec or VelMaxX or VelMaxY whichever is lower.
GetAutoGuiderVel ?	**KGvg**	VIEW_VEL	

? Indicates command supported through SkyWalker's Static Status Change Notification (SSCN) feature

Command name	**Code**	**Parameter or return**	**Description**
		QuickKeys	
SetQuickKey1Function	**KSq1**	<see below>	
GetQuickKey1_Function ?	**KGq1**	<see below>	
SetQuickKey2Function	**KSq2**	<see below>	
GetQuickKey2_Function ?	**KGq2**	<see below>	
SetQuickKey3Function	**KSq3**	<see below>	
GetQuickKey3_Function ?	**KGq3**	<see below>	
SetQuickKey4Function	**KSq4**	<see below>	
GetQuickKey4_Function ?	**KGq4**	<see below>	
SetQuickKey5Function	**KSq5**	<see below>	
GetQuickKey5_Function ?	**KGq5**	<see below>	
SetQuickKey6Function	**KSq6**	<see below>	
GetQuickKey6_Function ?	**KGq6**	<see below>	
SetQuickKey7Function	**KSq7**	<see below>	
GetQuickKey7_Function ?	**KGq7**	<see below>	
ExeQuickKeyFunction	**KEqk**	-->	Accepts a digit, 1-7.
For the above commands, the following assignment parameters apply:	"CalFromEnc", "CalFromLastGoTo", "DriftToggle", "FocusPulseToggle", "GoToAnchor", "GoToLast", "GoToPark", "Hunt", "InvertToggle", "Mark", "MarkReturn", "MirrorToggle", "Momentary1", "Momentary2", "Momentary3", "Nothing", "OneShot", "Return", "TimeStamp", "Toggle1", "Toggle2", "Toggle3" or "Wobble"		
? Indicates command supported through SkyWalker's Static Status Change Notification (SSCN) feature			

3.11. Time tools

Command name	**Code**	**Parameter or return**	**Description**
		Alarm	
SetAlarmOn	**LSao**	FLAG	String "Yes" to turn-on and "No" to turn-off SkyWalker's Alarm
GetAlarmOn ?	**LGao**	FLAG	

GetAlarmRemaining	**LGar**	HOURS	
SetAlarmTime0	**LSa0**	HOURS	Sets Alarm expiration time
GetAlarmTime0 ?	**LGa0**	HOURS	
SetAlarmWarningTime	**LSaw**	HOURS	This sets the time, ahead of the Alarm expiration time, when a warning occurs.
GetAlarmWarningTime ?	**LGaw**	HOURS	
SetAlarmWarningEnabled	**LSae**	FLAG	
GetAlarmWarningEnabled ?	**LGae**	FLAG	

? Indicates command supported through SkyWalker's Static Status Change Notification (SSCN) feature

Command name	**Code**	**Parameter or return**	**Description**
		Timer	
SetTimerOn	**LSto**	FLAG	String "Yes" to turn-on and "No" to turn-off SkyWalker's Timer
GetTimerOn ?	**LGto**	FLAG	
GetTimerRemaining	**LGtr**	HOURS	
SetTimerPeriod	**LStp**	HOURS	Sets Timer duration from when first turned-on to when it will expire
GetTimerPeriod ?	**LGtp**	HOURS	
GetTimerTime0 ?	**LGt0**	HOURS	Time at which the Timer will expire
SetTimerWarningTime	**LStw**	HOURS	This sets the time, ahead of the Timer expiration time, when a warning occurs.
GetTimerWarningTime ?	**LGtw**	HOURS	
SetTimerWarningEnabled	**LSte**	FLAG	
GetTimerWarningEnabled ?	**LGte**	FLAG	

? Indicates command supported through SkyWalker's Static Status Change Notification (SSCN) feature

Command name	**Code**	**Parameter or return**	**Description**
		TimeStamp	
SetTimeStampNow	**LSts**	<none>	SkyWalker sets TimeStamp2 to the value of TimeStamp1 and records the present time as TimeStamp1
GetTimeStamp1 ?	**LGt1**	-->	Returns "XX>HH:MM:SS", where XX is Time Stamp Format ("UTC" or "Local")

GetTimeStamp2 ?	**LGt2**	-->	Returns "XX>HH:MM:SS", where XX is Time Stamp Format ("UTC" or "Local")
GetTimeStampDelta ?	**LGtd**	HOURS	Returns the difference between TimeStamp1 and TimeStamp2
SetTimeStampFormat	**LStf**	-->	UTC or "Local"
GetTimeStampFormat ?	**LGtf**	-->	UTC or "Local"
? Indicates command supported through SkyWalker's Static Status Change Notification (SSCN) feature			

Command name	**Code**	**Parameter or return**	**Description**
			Time tools aggregate
GetTimeToolAggregate	**LGta**	-->	Returns AlarmRemaining, TimerRemaining in the format: "hh:mm:ss|hh:mm:ss". Note: the Client can use AlarmOn and TimerOn to determine if it is necessary to issue this aggregate.

3.12. Motor and gearing commands

Command name	**Code**	**Parameter or return**	**Description**
			Motor settings
SetMotorPolarityX	**MSpx**	SIGN	String "Positive" or "Negative"
GetMotorPolarityX ?	**MGpx**	SIGN	Returns "Positive" or "Negative"
SetMotorPolarityY	**MSpy**	SIGN	String "Positive" or "Negative"
GetMotorPolarityY ?	**MGpy**	SIGN	Returns "Positive" or "Negative"
SetMotorY_PolarityValidIn	**MSyv**	-->	String "InEast" or "InWest"
GetMotorY_PolarityValidIn ?	**MGyv**	-->	Returns "InEast" or "InWest"
SetMotorStepsPerRevX	**MSsx**	NN,NNN	Min of 1,000 max of 99,999,998
GetMotorStepsPerRevX ?	**MGsx**	NN,NNN	
SetMotorStepsPerRevY	**MSsy**	NN,NNN	Min of 1,000 max of 99,999,998
GetMotorStepsPerRevY ?	**MGsy**	NN,NNN	
? Indicates command supported through SkyWalker's Static Status Change Notification (SSCN) feature			

Command name	**Code**	**Parameter or return**	**Description**
Motor status & diagnostics			
GetMotorVelX	**MGvx**	-->	Returns current motor velocity in the format "sNN,NNNsteps/sec"
GetMotorVelY	**MGvy**	-->	Returns current motor velocity in the format "sNN,NNNsteps/sec"
GetMotorLoadX	**MGlx**		
GetMotorLoadY	**MGly**		
GetMotorDiagPositionX	**MGdx**	sNN,NNN	Offset, in steps, since last ZeroMotorDiagPositionX command
ZeroMotorDiagPositionX	**MZdx**	<none>	Zeros (resets) the MotorDiagPositionX counter
GetMotorDiagPositionY	**MGdy**	sNN,NNN	Offset, in steps, since last ZeroMotorDiagPositionX command
ZeroMotorDiagPositionY	**MZdy**	<none>	Zeros (resets) the MotorDiagPositionY counter

Command name	**Code**	**Parameter or return**	**Description**
Gear periodic error correction settings and status			
SetPECorrEnabled	**MSpe**	FLAG	Enables/disables periodic error correction
GetPECorrEnabled ?	**MGpe**	FLAG	
GetPECorrTrained ?	**MGpt**	FLAG	Returns "Yes" if periodic error correction is trained, "No" otherwise.
GetPECorrAppliedX	**MGcx**	sNN,NNN	Returns present magnitude of X-axis periodic error correction in the range of +32,767/-32,768.
? Indicates command supported through SkyWalker's Static Status Change Notification (SSCN) feature			

Command name	**Code**	**Parameter or return**	**Description**
Gear periodic error correction training			

StartPECorrTraining	**MSpt**	<none>	Starts PEC training if the following conditions are met: - Mount is polar aligned - ViewVel1 is specified - ViewVel1 is less than 15 arcsec/sec (so as not to cause RA backlash while training) - Alignment is complete - TrackRate is Sidereal - MoveMode is Track - ViewMode is PanScope (so that keyed movement is along telescope axes) - GearPeriod is not too large for PEC
CancelPECorrTraining	**MCpt**	<none>	Cancels PEC training
GetPECorrTrainingTimeRemaining	**MGtt**	SHORT_TIME	
GetPECorrTrainedDrift	**MGtd**	sD.DD	Arcseconds
GetPECorrTrainedMagPeakToPeak	**MGtm**	sD.DD	Arcseconds
GetPECorrTrainedMagRMS	**MGtr**	D.DD	Arcseconds
SavePECorrTraining	**MScb**	<none>	PEC corrections are saved to SkyWalker's Flash memory.
? Indicates command supported through SkyWalker's Static Status Change Notification (SSCN) feature			

Command name	**Code**	**Parameter or return**	**Description**
Gear periodic error measurement			
StartPEMeasurement	**MSpm**	<none>	Starts periodic error measurement if the following conditions are met: - Mount is polar aligned - ViewVel1 is specified - ViewVel1 is less than 15 arcsec/sec (so as not to cause RA backlash while training) - Alignment is complete - TrackRate is Sidereal - MoveMode is Track - ViewMode is PanScope (so that keyed movement is along telescope axes) - GearPeriod is not too large for PEC
CancelPEMeasurement	**MCpm**	None	Cancels periodic error measurement
GetPEMeasurementTimeRemaining	**MGmt**	SHORT_TIME	
GetPEMeasurementDrift	**MGmd**	sD.DD	Arcseconds
GetPEMeasuredPeakToPeak	**MGmm**	sD.DD	Arcseconds
GetPEMeasuredRMS	**MGmr**	D.DD	Arcseconds

? Indicates command supported through SkyWalker's Static Status Change Notification (SSCN) feature

Command name	**Code**	**Parameter or return**	**Description**
		Gear backlash error correction	
SetBLashEnabledX	**MSbx**	FLAG	Turns-on ("Yes") or off ("No") backlash correction for the X axis
SetBLashEnabledY	**MSby**	FLAG	Turns-on ("Yes") or off ("No") backlash correction for the Y axis
GetBLashEnabledX ?	**MGbx**	FLAG	
GetBLashEnabledY ?	**MGby**	FLAG	
SetBLashBurstMaxVelX	**MSrx**	-->	Velocity that backlash correction is applied in X: between "1steps/sec" to "30,000steps/sec"
SetBLashBurstMaxVelY	**MSry**	-->	Velocity that backlash correction is applied in Y: between "1steps/sec" to "30,000steps/sec"
GetBLashBurstMaxVelX ?	**MGrx**		
GetBLashBurstMaxVelY ?	**MGry**		
SetBLashCorrMagX	**MSmx**	NN,NNNsteps	Magnitude of backlash correction applied in X: between 1 and 30,000
SetBLashCorrMagY	**MSmy**	NN,NNNsteps	Magnitude of backlash correction applied in Y: between 1 and 30,000
GetBLashCorrMagX ?	**MGmx**	NN,NNNsteps	
GetBLashCorrMagY ?	**MGmy**	NN,NNNsteps	
GetBLashCorrAppliedX	**MGfx**	NN,NNNsteps	The present backlash correction applied in X. Listed as number of motor steps.
GetBLashCorrAppliedY	**MGfy**	NN,NNNsteps	The present backlash correction applied in Y. Listed as number of motor steps.

? Indicates command supported through SkyWalker's Static Status Change Notification (SSCN) feature

Command name	**Code**	**Parameter or return**	**Description**
		Gear Period	
SetGearPeriodX	**MSix**	NN,NNN	Number of motor steps in X to rotate the gear once. Min of 100 max of 99,999,998
GetGearPeriodX ?	**MGix**	NN,NNN	Number of motor steps in X to rotate the gear once. Min of 100 max of 99,999,998
GetGearCountX	**MGux**	NN,NNNsteps	Gear position in X in steps (measured from last GearIndex detection)

? Indicates command supported through SkyWalker's Static Status Change Notification (SSCN) feature

Command name	**Code**	**Parameter or return**	**Description**
		Gear Index	
SetGearIndexEnabledX	**MSex**	FLAG	Enable/disable use of Gear Index, on X axis, to zero GearCount once per gear rotation.
GetGearIndexEnabledX ?	**MGex**	FLAG	
GetGearIndexToggledX	**MGax**	FLAG	Toggles every time a GearIndex detection occurs on the X axis.
GetGearIndexSincePowerUpX	**MGox**	FLAG	Indicates if a GearIndex has been detected since power-up on the X axis.
SetGearIndexHysteresisX	**MShx**	NN,NNNsteps	X-axis offset between GearIndex position moving positive vs. moving negative.
GetGearIndexHysteresisX ?	**MGhx**	NN,NNNsteps	
GetGearIndexHysteresisCountX	**MGnx**	NN,NNNsteps	Measured/diagnostics X-axis offset between GearIndex position moving positive vs. moving negative.
SetGearIndexGateEnabledX	**MSgx**	FLAG	Enables use of gating signal (if SkyWalker model supports it) which is logically-ANDed to the GearIndex signal.
GetGearIndexGateEnabledX ?	**MGgx**	FLAG	
GetGearIndexGateStateX	**MGjx**	FLAG	Reports the state of the GearIndexGate input (if SkyWalker model supports it).
? Indicates command supported through SkyWalker's Static Status Change Notification (SSCN) feature			

Command name	**Code**	**Parameter or return**	**Description**
		Motor and Gearing aggregates	
GetMotorStatAggregate	**MGsa**	-->	Returns Motor Diag PositionX, Motor Diag PositionY, Motor VelX, Motor VelY, Motor LoadX, Motor LoadY in the format "XXXXXX|XXXXXX|XXsteps/sec|XXsteps/sec|XXX|XXX".
GetPEC_BacklashStatAggregate	**MGpa**		Returns PECorrAppliedX, PECorrAppliedY, BLashAppliedX, BLashAppliedY in the format "XXXXsteps|XXXXsteps|XXXXsteps|XXXXsteps".
GetGearStatAggregateX	**MGxa**		Returns Gear CountX, Gear Index ToggleX, Gear Index Gate StateX, Gear Index Hysteresis CountX and Gear Index Since PowerUpX in the format "XXXXsteps|FLAG|FLAG|XXXXsteps|FLAG".
? Indicates command supported through SkyWalker's Static Status Change Notification (SSCN) feature			

…For the remainder of the command reference see SW_Client.txt…

4. Special topics

4.1. Client's Alignment Wizard

It is recommended that the process of alignment be accommodated in Client software through the use of an alignment wizard. Possible logic for this alignment wizard is described here. Note: this logic is recommended because it is consistent with the logic used in SkyWalker's *Instrument Display* interface.

Step	Client action
1	User clicks the **Align from Star(s)** button. Pop-up the Alignment Wizard dialog box. It is recommended that the **CheckSiteTimeDateSetOnce** command be used, at this point, to verify that SkyWalker's site settings, time and date have been set at least once. This will avoid the inconvenience of the alignment being prohibited (if site, time and date have not each been set at least once) after the first sighting has been made.
2	Use the **GetAlignmentStartingPosition** command to determine what *Starting Position* is currently selected for use in SkyWalker.
3	If *Meridian Avoidance* is Full (found via the **GetMeridianAvoidMethod** command) **and** *Starting Position* is **None** then: Provide a screen (with suggested title "Choose side of sky") that allows the user to select between "In the East" or "In the West" for *Alignment Side*. Include instructive text (e.g. "In which side of the sky will alignment commence…"). Send the **SetAlignmentSide** command to SkyWalker with a parameter dependent on the user selection (i.e. "East" or "West"). Otherwise, it is unnecessary to prompt for the side of the sky that alignment will commence (see next step).

4	If *Starting Position* is **None** (regardless of *Meridian Avoidance*) then skip to step #6. Otherwise, provide a screen (with suggested title "Choose starting position") with a scrollable list of *Starting Positions* with the currently selected one as the default selection. Include instructive text (e.g. "Please verify that the telescope orientation is…"). If the user selects a different *Starting Position* then send a **SetAlignmentStartingPositionFull** or **SetAlignmentStartingPositionNonFull** command to SkyWalker to choose the new *Starting Position.* Note: if the user selects **None** for a *Starting Position*, and *Meridian Avoidance* is Full, then it is necessary to send the **SetAlignmentStartingPosition*** command with "None" as a parameter (previous paragraph) and go back to step #3 so that the user is prompted for the side of the sky to align from.
5	When the user clicks **Next,** send the **AlignFromStartingPosition** command to SkyWalker. At this point SkyWalker should have an *Alignment State* of **Preliminary** (unless they choose a *Starting Position* of "None" in the previous step, in which case SkyWalker will remain un-aligned for now).
6	Provide a screen (with suggested title "Choose alignment star") with a scrollable list of alignment stars. This list can either be consistent with SkyWalker's bright star list or the Client's own custom list. Include instructive text (e.g. "Select an alignment star from the list"). Use the ATCL object database commands to get this information if using stars consistent with SkyWalker's bright star list.
7	After the user clicks **Next,** provide a screen (with suggested title "Alignment star information") that shows important information on the star selected (e.g. Magnitude, Altitude, and Azimuth). Also on this screen, provide **GoTo** and **AlignOnStar** buttons.
8	If the user clicks **GoTo**, and the *Alignment State* is at least **Preliminary**, use the **GoToSelectedObject** or **GoToTargetRA_Dec** commands. This should slew the telescope near to the selected star. Note: as with any GoTo, the GoTo progress bar should be displayed while the GoTo is in progress.
9	If the user clicks the **AlignOnStar** button, send the **AlignFromSelectedObject** or **AlignFromTargetRA_Dec** command to SkyWalker. Note: if using the **AlignFromTargetRA_Dec** command, be sure to use the **SetEpochOfEntry**, **SetTargetRA** and **SetTargetDec** commands first.

10	If the *Alignment State* is still **Preliminary** then repeat from step #6 (since a second star is needed to complete alignment). Note: it might be useful to title the screen associated with the 2nd-pass through step #6 as "Choose second alignment star". If the *Alignment State* is **Aligned** then the Alignment Wizard is finished.

4.2. Instrument Display interface

4.2.1. Instrument Display commands

ATCL includes a sub-set of functionality that exposes SkyWalker *Instrument Display* interface. These are simply additional ATCL commands that allow Client software to implement an *Instrument Display* on their screen.

SkyWalker's *Instrument Display* interface is the only guaranteed fully implemented method to interface to all of SkyWalker's current features. There will be fancy Client programs (e.g. Maestro and even 3rd party) running under various operating systems however there is no certainty that they will always fully-support all of SkyWalker's features, nor stay up-to-date with enhancements to SkyWalker firmware, unless they implement an *Instrument Display* interface.

Clients can provide a dumb 8-line by 21 character display (compatible with Astrometric Instrument's HP2 handpaddle) with menu selection buttons and a back button and then, with little overhead code, expose SkyWalker's *Instrument Display*. The "engine" behind the *Instrument Display* resides on SkyWalker and ATCL provides a set of commands that signal *Instrument Display* button presses to SkyWalker and provide for SkyWalker to write the contents of the Client-hosted display.

In addition to the three return values possible with ordinary ATCL commands (i.e. ATCL_ACK, ATCL_NACK, and an actual return string), *Instrument Display* commands can return additional single special character responses. The complete list of possible *Instrument Display* return values is:

- Return string with semicolon termination character.
- ATCL_ACK
- ATCL_ID_CMND: a user-initiated action was successfully completed. The Client should issue a click sound.
- ATCL_ID_LINK: the user pressed a button that resulted in the display of a new *Menu Page* on the *Instrument Display*. The Client should issue a new-page sound.
- ATCL_ID_DING: the user pressed a button that does nothing. The Client should issue a ding sound.
- ATCL_NACK

In the table below, shaded commands are generally used only by the *Instrument Display* on a handpaddle (e.g. HP2) that has buttons to support the associated commands.

Command name	Mne-monic	Parameter(s)	Command description
ID_MenuButtonPressed	**IBpm**	Menu button #: “1” to “7”	A menu button was pressed
ID_Back	**IBbk**	None	The Back button was pressed
ID_Refresh	**IRfr**	None	Request that SkyWalker redraw the ID
ID_JumpMenuPage	**IJmp**	Menu page #: “0” to “99999”	Jump to menu page number
ID_JumpTop	**IJtp**	None	Jump to the top menu page
ID_JumpObjects	**IJob**	None	Jump to the Objects menu page
ID_JumpActions	**IJac**	None	Jump to the Actions menu page
ID_JumpStatus	**IJst**	None	Jump to the Status menu page
ID_JumpSettings	**IJse**	None	Jump to the Settings menu page
ID_JumpActionsQuickKeys	**IJqk**	None	Jump to the Actions/QuickKeys menu page
ID_Enter	**IBen**	None	The enter key was pressed
ID_NumButtonPressed	**IBpn**	Number button # “0” to “9”	A number button (keypad) was pressed
ID_CursorUp	**IBcu**	None	Move the cursor/selection up one
ID_CursorDown	**IBcd**	None	Move the cursor/selection down one
ID_CursorLeft	**IBcl**	None	Move the cursor/selection left one
ID_CursorRight	**IBcr**	None	Move the cursor/selection right one
ID_CursorPageUp	**IBpu**	None	Move the cursor/selection up one page
ID_CursorPageDown	**IBpd**	None	Move the cursor/selection down one page

ID_GetNextMapItem	**IGmi**	None	Get next item for a "tree view" Display Map. The return value is "Nnnnnnllxxxxxx". Encoding is as follows: Record Map item #nnnnn at indent level #ll as "xxxxxx" where: #nnnnn is the item number "00001" through "99999". Item number is incremented with sequential **IGmi** commands until there are no more items left, at which point the **IGmi** command return value is "Lnnnnn" where nnnnn indicates the last item number that SkyWalker provided. #ll is the indent level from "00" through "99". "00" implies no indent. "xxxxxx" is the NULL-terminated title of the map item (i.e. the Menu Page title). Example: assuming 300 items have already been processed, then: Main1 Child m1-1 Child m1-2 Main2 Child m2-1 Child m2-1-1 would be encoded as return values to sequential **IGmi** commands as follows: N0030100Main1 N0030201Child m1-1 N0030301Child m1-2 N0030400Main2 N0030501Child m2-1 N0030602Child m2-1-1 If that were the end of SkyWalker's list then L00306 would be returned in response to subsequent **IGmi** commands. See Note 1 for additional suggestion…

ID_ResetMapSequence	**IRms**	None	At power-up, SkyWalker resets its “map item” counter. The Client can reset it as well with the ID_ResetMapSequence command. This command should be used if the Client detects a communications fault.
ID_GetSelectedMapItem	**IGsi**	None	Retrieves the map item number for the Menu page that SkyWalker’s Instrument Display is currently “displaying”.
ID_EditBoxReturn	**IEbr**	Edit box contents	Returns the string currently in the Edit box to SkyWalker.
ID_EditBoxRoll	**IEbl**	Edit box contents	Returns the string currently in the Edit box to SkyWalker. SkyWalker then “rolls” the display units and re-issues the Edit box contents to the Client via the “E” Instrument Display asynchronous response (next section).

Note 1: when using the ID_GetNextMapItem command the Client will receive a complete list of all Menu Page titles, **that are not hidden.** Hidden Menu Pages exist and cannot be displayed on the Display Map since they don’t link from just one “parent” Menu Page. It is suggested that if the Client receives a Menu Page number (using the ID_GetSelectedMapItem command), that was not part of the list acquired with the ID_GetNextMapItem command (i.e. that is a “hidden” page), that the proper course of action is simply to display the last visited non-hidden Display Map node as yellow.

4.2.2. Instrument Display asynchronous responses

When SkyWalker needs to change the contents of the Client’s *Instrument Display*, it sends an asynchronous packet preceded by the ATCL_IDC_ASYNCH special character. The packet contains a return-command that the Client must act upon to assure that the contents of the *Instrument Display* remain up to date. Note: the string “xxxxx” in the below table is of variable length.

Packet string	Necessary action
C;	Clear the *Instrument Display*
Ofxxxxx;	Overwrite field text in field #f with characters “xxxxx”. ‘f’ is in the range of 0 through 7 (8 fields). If any character in “xxxxx” is preceded by 08h it is drawn in reverse video. Note: “xxxxx” should fully overwrite text in field #f. No original text should be left in the field even if “xxxxx” is shorter than what was in the field.
Bu;	Menu button is “live”: menu buttons 1-7 correspond to bits 0-6 in ‘u’. Note: bit 7 of ‘u’ is undefined. It may be 0 or it may be 1 so as not to cause ‘u’ to ever be NULL. Note: there is no menu button 0. There is a field 0, it is the top field of the display (for which there is no button).
Un;	Scroll up n lines (i.e. move field x to field x+n, if possible, where x is between 1 & 7)

Dn;	Scroll down n lines (i.e. move field x to field x-n, if possible, where x is between 1 & 7)
Exxxxx;	Fill edit-box field with "xxxxx".
Rxxxxx;	Fill edit-box field with "xxxxx" and provide units "rolling" button.
S;	Draw scroll-up/down and page-up/down buttons adjacent to the *Instrument Display*.
X;	Remove/hide any Edit box, page-up/down buttons or scroll-up/down buttons from the Client's screen (i.e. SkyWalker is done with them).

4.2.3. The Instrument Display edit-box

To change settings or cause action, general-purpose data must often be provided to SkyWalker. Examples include "12:23:45" for the RA coordinate of a GoTo, "14,440,000" for the motor steps per rev, or "72:43:23W" for the site longitude. ATCL provides one instrument display command and one asynchronous reply that allows the Client to enter/edit general-purpose data:

- SkyWalker will send the "Exxxxx;" or "Rxxxxx;" asynchronous reply (preceded by the ATCL_IDC_ASYNCH special character) to the Client whenever it needs the Client to place an edit-box on its *Instrument Display*. The "xxxxx" field of this reply provides the default contents for the edit-box. The Client must provide a means for the user to change the edit-box contents.
- Once the user has finished with the edit-box entry/editing they would hit <Enter>. The Client would then send the command ID_EditBoxReturn, with the semicolon-terminated contents of the edit-box, to SkyWalker-Servo.

Note: although not strictly required, it is recommended that the edit-box be placed directly on the *Instrument Display* area of the Client's console/window. When an edit box is active SkyWalker may also have other *Instrument Display* fields active (at a minimum, field 0 contains the title of the current Display Page). Therefore, this specification sets the requirement that the edit-box does not obscure the first 5 lines of the *Instrument Display*. SkyWalker will restrict its simultaneous use of the *Instrument Display*, when an edit box is active, to fields 0-4.

4.3. ATCL support from ASCOM

Astrometric Instrument's Maestro ATCL Client software is ASCOM-compliant. In addition to support for the majority of ASCOM Telescope Server Interface Methods (routines), Maestro also provides support for an ASCOM Client to pass ATCL commands directly to SkyWalker. This support is implemented through the CommandString() ASCOM method.

To pass an ATCL command directly to SkyWalker, call the CommandString() method with a string parameter that is an ATCL command. The response string of the CommandString() function is the response from SkyWalker. For example, to set SkyWalker's ViewVel4 to 0.8deg/sec you could use the following VBA CommandString() code:

```
Set Scope = CreateObject( "Maestro.Telescope" )
Response = Scope.CommandString( "!KSv40.8deg/sec;" )
```