Lick 2.4m Telescope

Software User Manual

SUM-13948-1

EOS Technologies, Inc. • 3160 East Transcon Way, Suite 180 • Tucson AZ 85706 • USA
Tel: +1 520 624 6399 • Fax: +1 520 624 1906 • Web: www.eostech.com

DOCUMENT CONTROL

Issue: 1

Prepared:	Elwood Downey		Date: 12/18/2009
Checked:	Aaron Evers		Date: 12/18/2009
Approved:	Andrew Lowman		Date: 12/18/2009
Configured:	Rose Tharp		Date: 12/18/2009

Document Revisions

Issue	Date	Description	Prep	Chk	Appr
0	11/10/2009	Initial Draft Release	ECD		
1	12/18/2009	Add ServoSystem schema; describe EOSLogToAScii; describe ERP file format; home after loss of DIOP power; more TelescopeServer schema fields; add index; move StarCal to separate document.	ECD	AE	AL

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 INTRODUCTION

1.1 INTRODUCTION

This document describes the software delivered with the UCO/Lick APF 2.4m telescope.

1.2 SCOPE

This document describes the following items:

- software architecture overview
- runtime processes
- configuration files
- server messsages
- event logs
- software maintenance activities
- powerup and shutdown procedures.

1.3 CONFIGURATION

This manual is a designated controlled document under the EOS Quality System.

CHAPTER 2 QUICK START

2.1 STARTUP

To power up the telescope, proceed as follows:

1. Push any observatory Emergency Stop to insure there will be no unexpected motion.

2. Nudge the telescope manually to be sure it is free to rotate in Azimuth and Elevation. Also Insure it is in within its normal range of motion and not in an extreme limit condition.

3. Turn on power by flipping the main breaker to the Up position. The main breaker is a three-phase unit located near the bottom rear of the blue rack.

4. Slide out the KVM keyboard and screen drawers and tilt the screen at a comfortable viewing angle. Note the screen hinges at the front so it must be pulled up from the back side. The KVM drawer is located about midway up the front of the blue rack. Power on the KVM and enter the password which is 00000000 (eight zeros).

5. Pull out some of the mouse chord and move it around the pad to wake up the screen. Log into Windows with user name EOST password lick-51. Eventually you will see a Windows XP desktop with icons down the left and across the top. Find the green EOST-Start icon on the left end of the top row and double-click. You will see several entries appear briefly across the task bar at the bottom of the screen then shortly these will disappear. Double-click the EOST-Start icon a second time and this time they will remain.

6. Turn off the observatory E-Stop and enable normal dome operation. Monitor the telescope for any motion. If no motions are observed, proceed.

7. Find the blue-and-white Telescope Interface icon on the right end of the top row and double-click. Open the Engineering tab.

8. Confirm that the time, shown in the right column, is correct.

9. Click the 'Home All' button on Engineering panel of the Telescope Interface to command all axes to Home. Watch for their respective Homed lights to come green and the overall telescope Status to become Homed.

10. If all axes home successfully, the telescope is ready for normal operation.

2.2 SHUTDOWN

To power down the telescope, proceed as follows:

1. Log on to the TCC, user name EOST password lick-51.

2. Open the Telescope Interface and maneuver the telescope into the desired park position.

3. Open the System Management Interface and stop all servers running on the TCC.

4. Shut down Windows.

5. At the rear of the blue rack, locate the three-phase breaker near the bottom and flip it down. The telescope is now completely powered down.

2.3 DIOP POWER FAILURE

If for any reason the DIO planar in the cabinet loses power or is accidentally switched off, all axes must be Homed again before resuming operation. This must be performed by operator initiative, the system does not know and hence can not annunciate that the home locations have become invalid.

CHAPTER 3 SYSTEM ARCHITECTURE

3.1 COMPUTER CONNECTIONS

The main computer is called TCC, Telescope Control Computer, a commercial grade rack mount configuration running Windows XP. See Figure 1 for an indication of how the software and hardware components are related. A Delta-Tau PMAC Turbo card, plus one Auxiliary IO card, are used to control the telescope motion electronics.

TCC cabling is managed through additional rack mounted interface boxes. The Digital I/O Planar, DIOP, contains signal conditioning and routing logic for cable connections to the EStop switch, M2 tilt and travel fail safe switches, mirror cover controller and the GPS time sync signals. The two Encoder Planars, ENCP0 and ENCP1, provides connections to the drive amplifiers, encoders and travel limit switches for the remaining telescope axes.

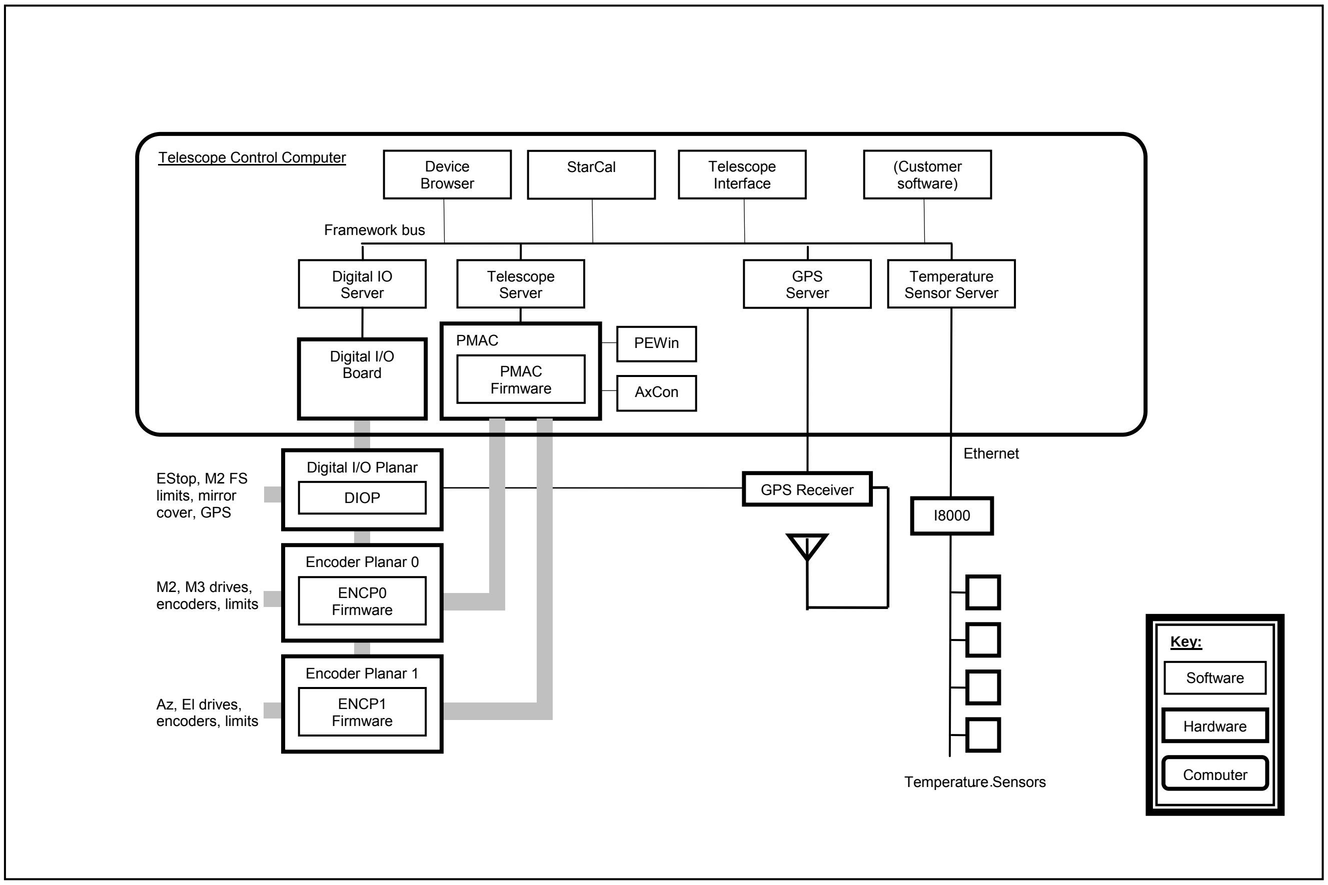


Figure 1. System overview

The heart of the system is a set of host processes that communicate through a well-defined interprocess communications scheme known as the Framework.

The Framework provides four servers. The DIO server is a thin interface over a general purpose I/O board. The telescope server operates all the motion axes on the telescope for slewing and tracking using the PMAC motion controller. The GPS server connects to a GPS receiver and provides accurate timing information. The Temperature server connects to a collection of RTDs placed strategically on the telescope structure.

PEWin and AxCon are tools that used for very low-level engineering purposes such as device checkout and servo tuning. Device Browser is a generic tool that is part of the Framework that can interface directly with any of the servers. These tools are not used for normal operation of the telescope. StarCal is a high-level tool used for managing pointing models. It can analyze an existing or candidate model, and take and install a new model. It is used occasionally whenever the pointing model might need updating such as after any mechanical or optical adjustments have been made to the telescope. It is not used for normal observing. Telescope Interface is the normal tool for operating the telescope in astronomical terms.

CHAPTER 4 SOFTWARE COMPONENTS

This section discusses each of the software processes involved in the telescope control system. Most are processes running on the TCC but we also discuss the firmware running on the PMAC motion controller. Some processes are part of the core system, some are low-level tools that are used primarily for engineering purposes, some are higher-level tools that are used more often by astronomers.

The discussions that follow assume the reader is familiar with the “EOS Space Systems Control System Client API Software User Manual”. In particular, a good understanding of the Device Meta Language is required. DML is a flexible, structured, text file format that captures all of the functions available for a particular framework server in a clear, hierarchical format. It builds on Integer, Float and String basic data types to form arrays and structures, and compound combinations of these, to describe each parameter or action as a member of a total schema. Each element in a nested DML definition uses dotted notation to uniquely traverse the schema hierarchy down to a particular element. A file containing an instance of DML has the suffix *.dmx* or *.dms*. There are usually one of each for each Framrwork server. A .dmx file is a configuration file for a server. A .dms file defines the Framework communications messages supported by a server.

4.1 PMAC FIRMWARE

The Turbo PMAC from Delta-Tau is a high performance multi-axis motion controller. It is used to operate each motor-encoder pair under servo control to drive each principle axis of the telescope in a precision manner.

The PMAC architecture is fundamentally based on the idea of registers and axes. There are hundreds of registers, the values of which determine everything from encoder scaling to the state of a limit switch. Reading and writing these registers allows states and values to be monitored or changed to affect desired operations of the motion axes.

These registers are available to programs written on board to the PMAC hardware, and some are available to the host computer via a shared memory port on the PCI bus. In this way, programs can be written on the PMAC that cooperate intimately with programs written on the host.

EOST uses 13 axes for the Lick APF, as shown in Table 1. The axis numbers are not contiguous because they also account for the encoders, with some axes using more than one encoder. Note PMAC Axis 8 is used to read 10 MHz time pulses, as if it were counting encoder values. This gives the PMAC direct access to a precision time reference.

Table 1 PMAC Axis Assignment

Axis	Duty

Axis	Duty
1	M2 Piston A
2	M2 Piston B
3	M2 Piston C
7	M3 Rotator
8	Clock
9	Azimuth
13	Elevation

Programs on board the PMAC itself are called PLCs, Program Logic Controls. Table 2 lists the PLCs written by EOST for the Lick APF. PLCs 0 and 2-6 are always running, the others are used only on demand. PLC 0 is the highest priority function of the PMAC, aside from the servo control loops themselves, so it is used to capture a nearly perfectly commensurate set of encoder data for reporting positions of each axis. PLC 1 is used to coordinate the initialization of the other PLCs. When the telescope system is started, PLC 1 is started, which starts the other required PLCs then turns itself off.

Table 2 PMAC PLC Functions

PLC	Always	Function
0	Yes	Sample axis positions atomically
1	No	Master reset
2	Yes	Master control
3	Yes	Detect faults and process errors
4	Yes	Manage jog mode for any axis
5	Yes	Manage tracking mode
6	Yes	Remote control of tuning parameters
10	No	Reset real-time clock
11	No	Manage homing of azimuth axis
12	No	Manage homing of elevation axis
13	No	Manage homing of M2 Strut A
14	No	Manage homing of M2 Strut B
15	No	Manage homing of M2 Strut C
16	No	Manage homing of M3 rotator axis

4.2 CORE FRAMEWORK SERVER PROCESSES

These processes are central to system operation and must be running for the telescope to function.

All required processes may be started or stopped using the convenient scripts attached to the shortcut icons located on the desktop labeled EOST Start and EOST Stop, respectively.

4.2.1 DIGITIAL I/O SERVER

The Digital I/O Server (DIOServer) is a gateway application that adapts the digital input and output signals from the Digital IO Planar (DIOP) into DML structures suitable for publication and manipulation by EOS Framework client applications.

The DIOServer is useful for monitoring telescope status flags and faults that are routed through the DIOP, such as power supply faults or failsafe limit switch orientation. The DIOServer is also useful for operating and monitoring the status of DIOP controlled subsystems, such as the Primary Mirror Covers.

To communicate with the DIOP, the DIOServer uses a Measurement Computing PC-DIO-96 general purpose I/O PCI card. It is configured so that 8 bits send information from the computer to the DIOP and 56 bits send data from the DIOP to the computer. The DIOServer must be run on a computer where the card and the appropriate driver are installed. The driver should be installed and tested using the vendor's test application first in order for the DIOServer to function properly.

The bits coming from the computer are basically the requests to open or close the mirror cover. Bits going to the computer are basically status indicators such as fail safe limits, power supply faults, mirror cover state and dome status including limit switches and emergency stop.

The DIOServer also has a simulation mode that does not require the interface card. In simulation mode the server does not need to be run on the computer where the driver or interface card is installed. The simulation mode allows developers to test the communications between the DIOServer and client applications. The simulation mode does not simulate the behavior of the DIOP - therefore it does not simulate status changes, characteristic of the true system.

Appendix C and Appendix D contain the configuration and schema files for the DIO Server as of this writing. They are included to facilitate self study from this manual and may not reflect the exact files as delivered in the final system. The real configuration and schema files are on the dome computer in c:\eos\bin\servers\ConfigServer\ConfigFiles\Telescope.

4.2.1.1 Configuration and Setup

The DIOServer uses standard DMX configuration files. Some familiarity with the EOS DMX configuration system is assumed. The DIOServer must be restarted to see any changes made to the configuration file. Editing configuration should be performed only by system engineers or administrators; however the configuration file may contain useful information for operators.

4.2.1.1.1 Application

The DIOServer supports the standardized *ApplicationConfig* structure, common to most EOST servers. This allows administrators to set the following common parameters:

- *Simulate* (boolean) - This allows administrators to enable or disable the simulation mode via the configuration file. This can also be enabled when running DIOServer via a command line argument. This is typically set to false.
- *Trace* (boolean) - This allows administrators to enable or disable the trace mode via the configuration file. This can also be enabled using the DML command *DeviceServer.Commands.DebugTrace* { } and via a command line argument. This turns on additional trace messages recorded in the server's log file. These are useful for debugging purposes. Due to the number of messages generated by this mode, it is not recommended that this be enabled during normal operation.
- *ScanRate* (float: seconds) - This allows administrators to adjust the duty cycle period that the server operates at. This may be adjusted as necessary to detect telescope signals, but performance of the server and CPU usage may be adversely affected if the period is too slow or too fast. The typical setting is 1 second.

4.2.1.1.2 Hardware

The hardware section of the configuration file allows an administrator to identify the DIO card being used by the server. The hardware section implements a *HardwareConfig* structure, which has the following parameters:

- *Type* (string: PCI-DIO-96 or PCI-DIO96) - This must match the string that is used to identify the card on the PCI bus. Supported National Instruments cards identify themselves as PCI-DIO-96. Supported Measurement Computing cards identify themselves as PCI-DIO96. The string is not case sensitive. The server will choose the appropriate driver to communicate with the card, based on this string and the host operating system. Note that the numbering used to identify each digital signal line varies by card, i.e. the port-to-pin mapping depends on the card being used.
- *Number* (integer) - The zero based number that identifies the card on the PCI bus. This is typically 0 if there is only one DIO card installed in the host computer; however this is assigned by the driver and may vary. The driver software should include a utility for setting or determining this number.

4.2.1.1.3 Devices

The DIOServer uses an array of logical device structures for separating and grouping various DIOP functions. The number of devices and the signals that are controlled by each device is entirely configurable. The example configuration has three logical devices named: *TelescopeStatus*, *TelescopeFaults*, and *MirrorCovers*. Each logical device is an implementation of a *DeviceConfig* structure, which has the following parameters:

- *Name* (string) - The logical name of the device. This is a uniquely identifiable string, conforming to DML syntax. The case is not sensitive. A corresponding device structure must be created in the Schema-DIOServer.dms file to provide access to the logical device to clients. An example logical device is the *MirrorCover* device which is used to operate and indicate the status of the primary mirror cover subsystem.
- *FieldName* (string) - The name of the corresponding top-level flag structure in the Schema-DIOServer.dms file. This schema structure must contain Boolean flags that correspond with each Input signal line that is to be reported to clients. The *FieldName* of the *MirrorCover* device is *MirrorCoverStruct*.
- *Inputs* (array of *InputConfig*) - An array of digital input signals associated with the logical device. Input signals are associated with status flags. Each input is an implementation of the *InputConfig* structure, which has the following parameters:
    - *Name* (string) - A human readable name for the digital signal bit. For example, the *MirrorCover* device reports the *MCOpenStatus*. The name must correspond to a boolean flag in the top-level flag structure for the device. An *MCOpenStatus* flag is located in the *MirrorCoverStruct* of the example schema file.
    - *Number* (integer) - The number which identifies the bit in the port-to-pin mapping. The *MCOpenStatus* example uses DIO port number 23.
    - *Invert* (boolean) - A true value will invert the signal as it is reported by the server. A false value reports the signal as it is read. This parameter is optional. The *FuseFaultStatus* in the *TelescopeFaults* device is the only inverted input signal in the example configuration.
    - *SimulatedInput* (boolean) - The input value to report for this bit when the server is in 'Simulate' mode. This parameter is optional. The *MCOpenStatus* example will always report TRUE while the server is simulating.
    - *LogPeriod* (float: seconds) - This value specifies the logging period for signal changes. This value is optional. Omitting this value will disable logging for this signal.
    - *FilterPeriod* (float: seconds) - This value specifies the filtering time constant for signal changes. This value is optional. Omitting this value will disable filtering for this signal.
- *Outputs* (array of *OutputConfig*) - An array of digital output commands associated with the logical device. Output commands may affect multiple digital output lines. Each

output is an implementation of the *OutputConfig* structure, which has the following parameters:

- *Name* (string) - A human readable name for the command. For example, the *MirrorCover* device has an *OpenMirrorCovers* command. Additionally, any command named Initialize will be called upon server startup. The name must correspond to a Command in the Schema file in order for operators to execute the command.
- *SetTrue* (array of integer) - A list of numbers that identify digital output signals, that will be set to true when an operator initiates this command. The *OpenMirrorCover* example sets the digital output numbered 52 to true. Bit 52 instructs the DIOP to open the mirror covers.
- *SetFalse* (array of integer) - A list of numbers that identify digital output signals, that will be set to false when an operator initiates this command. The *OpenMirrorCover* example sets the digital output numbered 54 to false. Bit 54 instructs the DIOP to close the mirror covers.
- *LogPeriod* (float: seconds) - This value specifies the logging period for the command. This value is optional. Omitting this value will disable logging for this signal.
- *Triggers* (array of *TriggerStruct*) - A list of trigger fields that specifies conditions that, when satisfied, will cause the command to be executed autonomously. The command will be executed if all of the conditions of any trigger are satisfied. This is useful, for example, to tell the DIOP to stop opening the mirror covers, once the mirror covers are already open. This avoids a conflict if the mirror covers are instructed to close using the hand-paddle. Each *TriggerStruct* has the following parameter:
    - *If* (array of integer) - A list of numbers that identifies digital input signals that must be conditionally satisfied to cause a trigger. Typically, the conditions must be true for any one trigger; however, the expected value can be inverted by negating the number. For example, to stop the DIOP from opening the mirror covers when they are already open, the *StopMirrorCovers* output command contains a trigger that will cause the command to execute if signals 23 and 40 are both true. If the list contained -24, the trigger will only execute the command if the DIOP is also not being command to close the covers. Note that the signal numbered 0 may be a valid signal, however it cannot be inverted.

4.2.1.2 Schema and Operation

The operation of the DIOServer as a citizen on the larger Framework is determined by the Schema-DIOServer.dms file. The schema file defines attributes and commands used to manipulate the DIO device via the Framework. Attributes are published structures containing status information and can be retrieved by clients either by subscription or by using a status retrieving command.

Note that, unlike most servers, the DIOServer's schema can be altered to accommodate changes to the server's configuration. This is useful, for example, if the DIOP is updated to control a new subsystem, the DIOServer can be configured to control the subsystem without the need to update the server itself. However, care must be taken to ensure that the schema and configuration files are synchronized and that such changes do not break other software, which relies on the schema of the DIOServer.

Commands and attributes are sent and received using the EOS Control System Client API. The Device Browser application allows access to this interface without needing to write a client application, however operators should rarely need to manipulate servers using the schema directly. Instead, they should use high-level user interfaces, such as the TelescopeInterface, which may present the attributes and commands in manner that does not resemble the interface described here. Operators should refer to the documentation of the client application.

4.2.1.2.1 ApplicationInfo

The *ApplicationInfoStruct* is a standardized attribute structure that is used by most EOST servers. It contains information about the DIOServer as it is running. The *ApplicationInfoStruct* has the following fields:

- *Version* (string) - A string containing the version number of the DIOServer. The current DIOServer will report "3.3.4.15" if using the 2.1.2 version of the EOS Framework.
- *BuildInfo* (string) - A string containing the date and time that the DIOServer was built. The date string will resemble: "Jun 30 2009 14:57:34."
- *SchemaVersionMajor* (integer) - A number representing the major schema version of the server. This will correspond with the X in the *__Schema_Version_Major__X* field of the *__Schema_Version__* structure if the server was built using the same schema as it is currently running with.
- *SchemaVersionMinor* (integer) - A number representing the minor schema version of the server. This will correspond with the Y in the *__Schema_Version_Minor__Y* field of the *__Schema_Version__* structure if the server was built using the same schema as it is currently running with.
- *Simulated* (boolean) - A flag that indicates if the server is running in 'Simulate' mode. The flag will be false if the server is operating normally.
- *Trace* (boolean) - A flag that indicates if the server is running with additional tracing turned on. If true, additional debugging information is being logged in the server's log file.
- *Fault* (*EventInfo*) - A structure that indicates why the server may currently be in a fault mode. The server will transition to 'Error' mode when a fault requiring operator intervention has occurred. The fault field will report information about the error that has occurred, however additional investigation, including examining log files may be required to determine the proper course of action. Sending the server the software Reset command will clear the fault field. The *EventInfo* structure contains the following fields:

  - Reason (string) - A string representing the likely cause of the error state.
  - Date (string) - A string representing the date and time the fault occurred.

4.2.1.2.2 HardwareInfo

The *HardwareInfoStruct* provides operators information about the DIO card being by the server. The following fields are reported:

- *Number* (integer) - The number of the board as configured in the configuration file.
- *Type* (string) - The type of board as configured in the configuration file.
- *DriverVersion* (string) - The version number of the driver that the server is using to interface with the DIO card. The string will be empty if the server is in simulate mode.

4.2.1.2.3 Commands

The DIOServer will respond to several common top-level commands that are common to most EOST servers. All command structures of servers written by EOST will contain two sub-structures: Parameters and Returns. The Parameters structure indicates the command parameters that may be required by the server. The Returns structure indicates the structure of the data that will be returned by the server upon successful operation. The following top level commands are for manipulating and retrieving information about the server independently from the logical device:

- *GetApplicationInfo* - Returns an *ApplicationInfo* structure, which matches the structure published in the Attributes section. This command provides an alternate method of retrieving the structure that does not require a subscription.
- *GetHardwareInfo* - Returns a *HardwareInfo* structure, which matches the structure published in the Attributes section. This command provides an alternate method of retrieving the structure that does not require a subscription.
- *Reset* - If the server is in an Error state, this command will attempt to reset the server and clear the error state.
- *DebugTrace* - This debug command allows an operator to enable or disable the trace mode of the server while it is running. This turns on additional logging that may be useful for debugging purposes. Debug commands may be removed on production installations.
- *DebugSetFault* - This debug command artificially instigates a server fault. This may be useful for debugging purposes. Debug commands may be removed on production installations.

4.2.1.2.4 Devices

DIOServer uses logical devices to group related functions. The devices described in the server's schema should correspond with the devices listed in the server's configuration. The example schema has three devices: *TelescopeStatus*, *TelescopeFaults*, and *MirrorCovers*.

Each device also has a structure indicating published attributes and a structure indicating the commands that can be used to operate the device. The attributes structure for each device must contain an instance of the top level structure, indicated by the device's *FieldName* parameter in the configuration file. This structure will contain a boolean flag that corresponds with the flags indicated in the device's *Input* list in the configuration file. For example, the *MirrorCovers* device has a *MirrorCoverStatus* instance called *Status*, which contains all of the flags associated with the operation of the mirror cover subsystem. The status structure can also be retrieved using the device's *GetStatus* command.

Each device also has commands for operating the device. Each command corresponds directly with an output described in the *Outputs* list of the configuration file. The server does not provide a mechanism for device commands to have any parameters; however the schema typically shows an empty parameter structure to be consistent with EOST standards. Similarly, device commands, besides *GetStatus*, do not return anything. The *MirrorCovers* device's *OpenMirrorCovers* is an example of a command that is used to operate the mirror cover subsystem.

4.2.1.3 Startup and Shutdown

Under normal operation, operators should not have to start and stop the DIOServer since it would be managed by the system management interface or scripts. If the DIOServer is started manually, the following command line arguments are available:

- /SIMULATE - Instructs the server to start in 'Simulate' mode, overriding the setting in the configuration file. This is useful for testing the communications with client software. However, the DIOP is not simulated, so status changes may not occur as they do on the real system.
- /TRACE - Instructs the server to start with 'Trace' mode enabled. This causes the server to log additional debugging information. This switch overrides the setting in the configuration file. The *DebugTrace* command may still be used to turn off trace mode after the server has started. Trace mode should not be enabled under normal operation.

4.2.1.3.1 Framework 2.1.x Parameters

The Framework 2.1.x version of the server may require the following additional parameters. Note that case and order must be preserved:

- -name <ServerName> - The <ServerName> field indicates the name of the server as it is registered with the Framework system. This is usually "DIOServer," however it may be changed if the system has been configured to register the server under a different name.
- -location <Location> - The <Location> field indicates the logical location of the server as it is registered with the Framework system. This is usually "Telescope," however it may be changed if the system has been configured to register the server with a different location.

- -isManaged <ManagedFlag> - The <ManageFlag> is either 'true' or 'false' and indicates if the server is being managed by the system management interface.

4.2.1.3.2 Framework 2.0.x 'Local' Configuration File

The Framework 2.0.x version of the server must have a 'local' configuration file in the same directory as the executable, with the following example contents:

```
LocalConfig
{
    Name      STRING
    Location  STRING
}

@

LocalConfig
{
    Name      "DIOServer"
    Location  "Telescope"
}
```

This file should have the same name as the server, but with a ".dmx" extension, i.e. DIOServer.dmx. This provides the Framework system with the same registration information as the -name and -location command line parameters do for the 2.1.x version of the Framework.

4.2.1.3.3 System Configuration Notice

The EOS Framework system is made up of a handful of communication, configuration, and operational servers. Several prerequisite servers must be running and properly configured on the observatory network for the DIOServer to detect and register itself with the system. Several of these system servers also require configuration to expect the registration attempts by the DIOServer. The configuration and operation of these servers is outside the scope of this document.

4.2.1.3.4 Shutdown

The DIOServer may be shutdown a number of ways. Typically it is shutdown by either the system management interface or by a shutdown script, which would shutdown the entire system. The DIOServer also responds to a 'Break' signal such as control-c if running in a console and will shutdown if such a signal is encountered. Operating system dependant 'kill' signals can be used to shutdown a background instance, assuming adequate permissions and access to the host computer.

4.2.2 TELESCOPE SERVER

The Telescope Server (TelescopeServer) is the heart and brain of the telescope. It accepts target definitions in high-level astronomical coordinates, applies pointing model corrections to arrive at encoder positions, and computes overlapping 50 ms tracks for the PMAC to follow. These tracks are updated sufficiently often to result in smooth continuous tracking of the defined target trajectory. The telescope server also performs the opposite function of reading the encoders, applying an inverse pointing model and reporting the current pointing direction of the telescope in astronomical coordinates.

The telescope consists of a collection of servo axes that control where the telescope optics are pointed. These include the gimbal axes (azimuth and elevation), the secondary focus (a set of three pistons) and the tertiary mirror (rotation and tip), and optionally a Nasmyth instrument rotator. There may be other servo axes in the telescope control system, such as the autoguider axes and the primary mirror actuators, but these are not considered part of the telescope and are not controlled by the commands in this section.

Telescope commands are used to control the position of the telescope and monitor its status.

At any time the telescope will be in one of four modes: Disabled, Homing, Tracking, or Stopped. The state of Parked is the same as being Stopped in a prescribed location.

In Disabled mode, power is removed from the telescope axes. The servo control system is disabled, and the telescope axes do not move. This is the default mode for the telescope when the telescope control system is started. In order to leave Disabled mode and start Tracking, or to hold a fixed position in Stopped mode, the telescope axes must first be Homed. This is a process that allows each axis to locate an absolute position reference so that its true position is known. Homing can either be done automatically by the telescope control system on receipt of a command to start tracking or to park the telescope, or it can be commanded explicitly by using the HomeAll command.

Tracking mode is the normal mode of operation when using the telescope. During tracking, the telescope follows a track that corresponds to a specified target. The target may be a celestial object moving at sidereal rate, an object in orbit moving at non-sidereal rate, a stationary position relative to the telescope, or any other conceivable motion. Target tracks are defined in terms of azimuth and elevation positions, hour angle and declination positions, or Right Ascension and Declination positions in epoch J2000.0. Simple tracks consist of just one or two points, but tracks can be made up of many points at any desired resolution.

While tracking, the secondary focus position is continually being adjusted to compensate for movement and flexure of the optical support structure as the temperature changes and the gravity vector varies with telescope elevation. Other corrections are also applied, including adjustment of the telescope position for atmospheric refraction and application of the mount model (pointing model). A client can adjust the telescope position for its own purposes with the

SetOffset message. The client also has control over several other characteristics of the telescope, such as the optional instrument rotator position, the secondary focus, the offset of the tertiary mirror, and selecting which instrument port to use.

When the telescope is stopped, it holds its current position under servo control. All adjustments and compensations are temporarily disabled.

The TelescopeServer also has a simulation mode that does not require the PMAC. The simulation mode allows developers to test the communications between the TelescopeServer and client applications. Simulation mode also provides a convenient way for users to learn the several client applications, such as Device Browser, Telescope Interface and StarCal, on a normal personal computer with no special hardware available.

The configuration and schema files are on the dome computer in c:\eos\bin\servers\ConfigServer\ConfigFiles\Telescope.

4.2.2.1 Configuration and Setup

The TelescopeServer uses standard DMX configuration files. Some familiarity with the EOS DMX configuration system is assumed. The TelescopeServer must be restarted to see any changes made to the configuration file. Editing configuration should be performed only by system engineers or administrators; however the configuration file may contain useful information for operators.

The main Configuration structure consists of the following sub components.

4.2.2.1.1 Application

The TelescopeServer supports the standardized *ApplicationConfig* structure, common to most EOST servers. This allows administrators to set the following common parameters:

- *Simulate* (boolean) - This allows administrators to enable or disable the simulation mode via the configuration file. This can also be enabled when running TelescopeServer via a command line argument. This is typically set to false.
- *Trace* (boolean) - This allows administrators to enable or disable the trace mode via the configuration file. This can also be enabled using the DML command *DeviceServer.Commands.DebugTrace* { } and via a command line argument. This turns on additional trace messages recorded in the server's log file. These are useful for debugging purposes. Due to the number of messages generated by this mode, it is not recommended that this be enabled during normal operation.
- *PublishPeriod* (float: seconds) - This allows administrators to adjust the rate at which clients who have asked to received published data on a periodic basis will receive such data by default.

- *TrackSafe* (bool) – This is a flag used for debugging code changes to the low level motion control algorithms. It forces the telescope to “track” a fixed location regardless of the commanded target motion. This flag must always be set False for normal operation.
- *EarthPolarRotationFile* (string) – This specifies the full Windows path name to the file containing earth polar rotation data.
- *TrackPeriod* (float: seconds) – The tracking system regularly computes a set of points for the servo system to follow. Due to limited memory the size of this set is limited. The system must provide a new set of points before the previous set are completed to maintain continuous operation. This parameter sets the duration of time for which the path is computed. It is a careful balance between computation resources and memory resources; it should not be changed under normal circumstances.
- *WarningEventPeriod* (float: seconds) – This sets the minimum elapsed time between successive logging of various timing related warnings. All of the timing events are automatically retried almost always successfully, so this saves log space since they tend to occur in brief bursts.
- *SetDataEventPeriod* (float: seconds) - This sets the minimum elapsed time between successive logging of setting new mets data, setting new truss temperature, setting new refraction values and setting offset values.

4.2.2.1.2 Site

This structure defines geographical location and default parameters of the telescope environment. It includes the following parameters:

- *GeodeticLatitude* (float: degrees north) – This is the common or geographic latitude of the telescope location. More precisely, it is the angle between the equatorial plane and a line normal to the reference ellipsoid through the telescope location.
- *Longitude* (float: degrees east) – This is the geographic longitude of the telescope location, measured eastward from the 0 longitude reference meridian.
- *Height* (float: meters) – This is the height of the telescope location above the reference ellipsoid, in meters. The value must be between -200 and 10,000 meters.
- *SunAvoidanceEnabled* (Boolean) – This indicates whether the telescope should not allow itself to be pointed near the sun. If this is True, the additional parameters *SolarHalo* (float: degrees) should be set to the minimum angular distance that will be allowed between the current sun position and the telescope pointing direction and *SolarHorizon* (float: degrees) should be set to the horizon elevation of the sun below which the avoidance algorithm will not be enforced. Note this feature does not perform look-ahead, *i.e.,* commands that will move the telescope into the forbidden zone will commence and then the telescope will stop as it enters the zone. Note also that sun avoidance is *not* enforced while the telescope is homing.

- *DefaultTemperature* (float: °C), *DefaultPressure* (float: millibars), *DefaultHumidity* (float: percent), *DefaultTrussTemperature* (float: °C) and *DefaultWavelength* (float: nanometers). These parameters set default values for several environmental parameters. They are only read once when TelescopeServer starts and remain in effect only until or unless further information becomes available.

4.2.2.1.3 Skyline

This structure is used to define a lower and upper horizon profile, below or above which the telescope will not move. Note the avoidance profiles are *not* enforced while the telescope is homing. It includes the following parameters:

- *SkyLineHysteresis* (float: degrees) – This is a soft tolerance above or below which the telescope is allowed to intrude into the lower or upper horizon profiles. Since the profile is implemented as a piecewise approximation to the ideal preferred horizon, this allows for some flexibility in the efficacy of the profile.

- *MaximumIntrusion* (float: degrees) – This is a hard limit above or below which the telescope should never intrude into the lower or upper horizon profiles. If the telescope finds itself within either profile more than this amount, it will stop and reject further commands to move. The only way to resume operation again is to re-home the telescope.

- *MinimumElevation* (array of SkylinePoint: degrees) and *MaximumElevation* (array of SkylinePoint: degrees) – These are tables of elevations at any specified azimuth values that define lower and upper horizon profiles. The elevation at an arbitrary azimuth is derived using linear interpolation between the nearest surrounding two entries.

4.2.2.1.4 TimeSource

This structure configures the time system. It includes the following parameters:

- *TimeServer* (string) – Network location of GPS time server.

- *MaximumSyncError* (float: seconds) – Maximum allowable difference between servo clock and time source.

- *MaximumPCSyncError* (float: seconds) – Maximum allowable difference between servo clock and PC clock.

- *FailuresIgnored* (integer) – Number of failed synchronization checks ignored before reporting an error.

- *MinimumValidYear* (integer) – Minimum allowed year reported by clock.

- *MaximumValidyear* (integer) – Maximum allowed year reported by clock.

- *ClockRetryPeriod* (float: seconds) – How often to retry synchronizing the clocks if it failed.

- *ClockCheckPeriod* (float: seconds) – How often to check that the clocks are synchronized.

- *ServerRefreshPeriod* (float: seconds) – How often to update the time from the time server.

4.2.2.1.5 InstrumentPort

This structure defines several parameters that together describe where an instrument is located and how it behaves in so far as it relates to telescope motions and offsets. It includes the following parameters:

- *Name* (string) – This is a brief name for the instrument position. It is the name that will be presented in the Telescope Interface to indicate this port is to be used.

- *InstrumentOrientation* (float: degrees) – This is the angle by which the direction to be considered “up” for the instrument on this port is rotated clockwise from zenith as viewed looking towards the tertiary mirror. Set correctly, this will result in a +Y tracking offset to move the image up on the instrument display.

- *MirrorImage* (boolean) – This indicates whether the instrument is considered to flip its image about the Y axis. Set correctly, this will result in a +X tracking offset to move the image towards the right on the instrument display.

- *FocusOffset* (float: millimeters) – This is an additional focus compensation that gets added for this instrument in the same way that values are added for temperature and elevation compensation and the value of the SecondaryFocus.ReferencePosition parameter. Set correctly, this will result in proper focus for this instrument when the focus position reported by Telescope Interface is zero.

- *PortSelectors* (array of PortSelectorConfig) – This defines how other axes of the telescope are to be positioned when this instrument port is selected. Typically this specifies a position for the Tertiary rotator.

- *MountModel* (string) – This specifies the name of the mount model that is automatically loaded when this instrument port is selected. If not specified, a default mount model is used.

4.2.2.1.6 SecondaryFocus

This structure describes the secondary focus mechanism and how it behaves.

The secondary for this telescope is mounted on a stage controlled by three linear actuators each located 120 degrees apart about and parallel to the optical axis. By making small changes to the lengths of these actuators, the plane of the secondary mirror can be changed in tip, tilt and focus.

For reference, these terms are defined as follows:

- *Tip* - rotation about an axis through the secondary mirror parallel to the elevation axis, *i.e.*, “up and down” when the telescope is pointing towards the horizon. A positive value represents the top of the secondary mirror moving away from the primary mirror. The angle refers to the shift in position as it would appear on sky.

- *Tilt* - rotation about an axis through the secondary mirror perpendicular to both the optical axis and the elevation axis, *i.e.*, “side to side” when the telescope is pointing towards the horizon. A positive value represents the right side of the secondary mirror moving away from the primary mirror, as viewed from the primary mirror looking towards the secondary mirror. The angle refers to the shift in position as it would appear on sky.

- *Focus* – motion along an axis connecting the centers of the primary and secondary mirrors, *i.e.*, towards or away from the primary mirror. A positive value represents a motion away from the primary mirror.

- *DecenterX* - motion in a plane perpendicular to the optical axis and parallel to the elevation axis, *i.e.*, “side to side” when the telescope is pointing towards the horizon. A positive value represents a motion to the right, as viewed from the primary mirror looking out towards the secondary mirror.

- *DecenterY* - motion in a plane perpendicular to the optical axis and perpendicular to the elevation axis, *i.e.*, “up and down” when the telescope is pointing towards the horizon. A positive value represents a motion moving up, as viewed from the primary mirror looking out towards the secondary mirror.

The structure includes the following fields:

- *Present* (boolean) – This field indicates whether the secondary mechanism is present. If set to False, the control system will ignore the mechanism entirely. This parameter is normally set to True.

- *Actuators* (array of SecondaryFocusActuator) – This is an array of three structures that defines the as-built position of each actuator relative to the optical axis passing through the secondary. Each structure contains the following fields:

  - *ServoAxis* (string) – This is the name by which the axis is known to other Framework clients.

  - *Angle* (float: degrees) – This is the angular position of the actuator axis, defined as positive clockwise from up as seen looking along the optical axis into space from the primary mirror.

  - *Radius* (float: millimeters) – This is the distance between the actuator axis and the optical axis passing through the secondary.

- *MirrorToImageTipRatio* (float: unitless) – This is the ratio of physical angular mirror tip rotation to the amount of angular "sky" motion it appears to produce on an image at the focal plane. Set in this way, tip commands sent to the secondary are input in terms of their effect on sky positions, and then the operator can easily command an opposite telescope offset in elevation to put the target back into its original position on the focal plane if desired. For optical testing purposes it can be useful to set this value to 1.0 so that tip commands to the secondary are in terms of physical mirror motion, but this is not the normal operating intent.

- *MirrorToImageTiltRatio* (float: unitless) - This is the ratio of physical angular mirror tilt rotation to the amount of angular "sky" motion it appears to produce on an image at the focal plane. Set in this way, tilt commands sent to the secondary are input in terms of their effect on sky positions, and then the operator can easily command an opposite telescope offset in azimuth to put the target back into its original position on the focal plane if desired. For optical testing purposes it can be useful to set this value to 1.0 so that tilt commands to the secondary are in terms of physical mirror motion, but this is not the normal operating intent.

- *FocusLimits, TipLimits, TiltLimits, DecenterXLimits, DecenterYLimits* (float) – These set the maximum range of motion for the secondary mount in the various directions. These are set at the factory and should not be modified.

- *ReferencePosition* (struct FocusPosition) – When the secondary mount is at these values, it is reported as being at 0 to the other Framework clients. SetFocus commands are relative to this location.

- *MinimumMoveSize* (struct FocusPosition) – This serves to document the smallest move that can be physically reliably made with the secondary mount system. Commands to move less than this will claim to succeed but will not in fact cause any motion to occur.

- *FocusElevationCompensationTable* (array of FocusElevationCompensationPoint) – This is a list of corrections that are added to the commanded secondary mount positions as a function of telescope elevation. Compensation values at intermediate elevations are interpolated from the table. The need for such corrections is due to sag in the spider that holds the secondary. As a rule, there are small effects to focus and tip but not to tilt since the gravity vector has no lever arm in that dimension to cause flexing.

- *FocusTemperatureCompensationtable* (array of FocusTemperatureCompensationPoint) – This is a list of corrections that are added to the commanded secondary mount positions as a function of telescope truss temperature. Compensation values at intermediate elevations are interpolated from the table. The need for such corrections is due to expansion and contraction of the truss arm steel when temperature changes. As a rule, there are small effects on focus but not tip or tilt, since the truss is essentially symmetric about the optical axis and thus all dimensional change is entirely along the optical axis.

- *FocusTargetRangeCompensationTable* (array of FocusTargetRangeCompensationPoint) – This is a list of corrections added to the secondary mount positions to accommodate targets that are closer than "infinity". Compensation values at intermediate ranges are interpolated from the table. The need for such corrections occurs because as the distance to a target moves closer to the telescope, the focal plane also moves closer to the primary mirror. Although the FocusTargetRangeCompensationPoint structure also includes members for tip, tilt and decentering, these corrections are typically all zero. Note that in the table, a TargetRange of zero denotes a target distance of infinity.

- *AzimuthTiltCompensationTable* (array of AzimuthTiltCompensationPoint) – This is a table of changes in azimuth that result from a change in secondary tilt, both in arc seconds. Usually this is set up for on-sky azimuth so that changes in pointing due to changes in tilt can be easily offset in azimuth. However, during optical alignment it can be useful to set these so tilt offsets can be considered in units of raw mirror angles; see *MirrorToImageTiltRatio*.

- *ElevationTipCompensationTable* (array of ElevationTipCompensationPoint) – This is a table of changes in elevation that result from a change in secondary tip, both in arc seconds. Usually this is set up for on-sky elevation so that changes in pointing due to changes in tip can be easily offset in elevation. However, during optical alignment it can be useful to set these so tip offsets can be considered in units of raw mirror angles; see *MirrorToImageTipRatio*.

4.2.2.1.7 ServoSystem

This structure specifies several parameters required to set up the communication between the servo system and PLC firmware running on the PMAC and also the parameters for operating and simulating each servo axis at the hardware level. Servo axes at this level are operated independently from any high level modeling, such as sidereal coordinates systems or mount models. Most of the parameters in this section are carefully chosen to correspond with the motor and encoder system hardware for each axis, and should not be changed unless the axis hardware is changed.

The structure includes the following fields:

- *PmacDevice* (integer) – This defines which PMAC controls this device. This should always be zero.

- *PmacRomVersion* (string) – This defines the version of the internal PMAC software. It must match what the PMAC reports.

- *PlcVersion* (string) – This defines the version of the PLC firmware installed in the PMAC written by EOST. It is in the form of *major.minor.* The Major number reported by the PMAC must match exactly what is expected by the software in the TCC. The Minor number need not match but will be logged if it does not.
- *Simulation* (struct *ServoSimulationConfig*) – This structure specifies some of the simulation parameters that affect all servo axes. The structure has the following parameters:
  - *PlcVersion* (string) – This string specifies the version of the servo firmware that will be reported by the simulated control system. If omitted, the value specified as the expected version will be reported as the simulated version.
  - *TrackLookahead* (integer) – This value specifies the number of look-ahead cycles to simulate during tracking. If omitted the value defaults to 3, which is the hard-coded value used by the server to control the number of track points that are pre-calculated and fed to the servo system ahead of time. Under simulation mode, this value affects the duration and accuracy of simulated tracks.
  - *MachineInputs* (string), MachineOutputs (stirng), DatInputs (string), and SelOutputs (string) – These parameters specify the default start-up state of the general-purpose digital inputs and outputs connected to the servo control system. Each parameter takes eight hexadecimal digits as characters, which are converted to bitwise flags, using the least significant bit first. These are typically all zeros, unless a specific fault condition or other status flag is being simulated.
- *ServoAxes* (array of ServoAxis) – This is a large structure containing many details of each axis under PMAC control. All of these fields are set by EOST and will rarely, if ever, need changing. However, if circumstances change such as if the weight loading on the telescope changes substantially, the tuning may require small changes. New tuning parameters may be specified in the array of ServoAxisTuning structs named Tuning. Any tuning parameter not specified in this dmx file will be set to the default value stored in the PMAC firmware. To explore new tuning parameters, it is recommended to use the AxCon tool (see §4.3.2.2). If, using AxCon, it is determined that one or more tuning parameters should be permanently changed, new values can be installed by editing the Default field of the ServoAxisTuning structure that has the matching tuning parameter Name. After saving the edited file, shut down and restart the TCC (see §Chapter 2) to be sure the new values remain as expected. Note that tuning parameters that are changed in this manner will only become active once the TelescopeServer is running; until then the original default values in the PMAC PLC firmware will still be in effect. The PMAC values are also used if there is no Default value at all in the corresponding dmx entry. Each axis is configured with the following parameters:
  - *Name* (string) – The name of the axis. This name is used to reference the axis by the higher level configuration, such as identifying the axis that operates the instrument field de-rotator. The name is also used to identify the axis to client software that allows diagnostic, status and discrete operation of the servo axis.
  - *MotorNumber* (integer) – The number that identifies the motor in the servo control system. This is determined by the hardware channel that the motor is connected to.

- *Encoders* (array of *ServoEncoderConfig*) – Identifies the encoder number and scale for each position and velocity encoder associated with the servo axis. The encoder number is determined by the hardware encoder channel that the encoder is connected to. The scale is the ratio of counts for the encoder to the number of counts for the axis. This is determined by the type of encoder, and the number of encoders attached to the axis.
- *Units* (string) – This field identifies the real-world base units that will be used for specifying several of the position parameters noted below. Valid units may be specified from the following list:
    - Radians
    - Degrees
    - Arcminutes
    - Arcseconds
    - Microradians
    - Hours
    - Revolutions
    - Meters
    - Metres
    - Millimeters
    - Millimetres
    - Microns
    - EncoderCounts
    - Encoder Counts
- *EncoderScale* (float: encoder counts per *Unit*) – The scaling factor to convert encoder counts into real-world units.
- *ReferencePosition* (float: encoder counts) – Position of the absolute reference point. This is the position where the axis will report a position of 0 in real-world units. All other positions reported will be relative to this position.
- *MinimumPosition* (float: encoder counts) – Minimum end of axis range. The software will not allow the axis to be commanded below this point. The value is also used to allow the software to control the deceleration of the axis prior reaching the limit. The value should correspond, or precede the minimum end-of-travel (EOT) limit switch for the axis. **Note:** This value is ignored during homing.
- *MaximumPosition* (float: encoder counts) – Maximum end of axis range. The software will not allow the axis to be commanded beyond this point. The value is also used to allow the software to control the deceleration of the axis prior reaching the limit. The value should correspond, or precede the maximum end-of-travel (EOT) limit switch for the axis. **Note:** This value is ignored during homing.
- *MaximumVelocity* (float: *Units* per second) and *MaximumAcceleration* (float: *Units* per second per second) – These identify the software velocity and acceleration limits. These values are determined based on the capabilities of the motor, amplifier, and

the safety and stability of the axis. **Note:** These settings should never be increased beyond the telescope specifications.

- *ParkPosition* (float: encoder counts) – The optimal parking position for the axis. The axis will be commanded to this position when the telescope is commanded to park. This position is usually set to minimize the duration of the homing procedure, by parking the axis near the first limit switch that the homing procedure expects to see or however the position may be set for operator convenience.
- *InstabilityLimit* (float: *Units* per second) – This setting enables stability monitoring on the axis. The stability monitoring system counts the number of times the displacement from the average velocity of the axis exceeds this limit for a specific sample period. The server will abort tracking if the count exceeds a value ultimately determined by the track period setting. A setting of zero disables stability monitoring for the axis.
- *HomeTimeout* (float: seconds) – Specifies the maximum time to allow for the axis to home. The telescope will report a homing failure if the axis does not home within the specified time. A value of zero disables the timeout and the telescope will wait indefinitely for the axis to complete homing.
- *MustHomeWith* (array of string) – Specifies any axes, by name, that must be homed simultaneously to this axis.
- *MustHomeAfter* (array of string) – Specifies any axes, by name, that must be homed prior to this axis.
- *AmplifierEnableDelay* (float: seconds) – Specifies an amount of time to wait after the axis amplifier has been enabled, before operating the axis. A value of zero disables the delay. If the value is omitted, the default delay of 0.1 seconds will be applied.
- *Tuning* (array of ServoAxisTuningConfig) – An optional array of servo tuning variables. If supported by the PLC firmware, this table must be populated with entries for each tuning parameter that may be updated dynamically. **Note:** Tuning parameters severely impact the stability and performance of the servo axis. Tuning commands should only be used by trained engineering personnel.

  Each tuning variable is specified using the following parameters:

  - *Name* (string) – The name of the tuning parameter. This name is used to identify the variable when using the SetTuning or GetTuning commands. The name must be unique for each tuning variable for each axis.
  - *Index* (integer) – The internal index for accessing the tuning variable. This index corresponds with the available tuning parameters in the firmware PLC that handles application of tuning parameters dynamically. The index may only be specified once for each servo axis.
  - *Min* (integer) and *Max* (integer) – The minimum and maximum valid values for the variable. Tuning values will be rejected if the specified value is less than the minimum or more than the maximum. The minimum and maximum values are determined by the servo control system and should not be altered.
  - *Default* (integer) – If specified, the named axis tuning variable will be initialized with this value and any subsequent ResetTuning commands will reset the tuning to this value. If omitted, the axis will use the default tuning defined in the PMAC

firmware. Subsequent ResetTuning commands will reset tuning to the default defined in the servo control firmware.

- *Simulation* (struct *SimulatedMotorConfig*) – This structure allows the specification of several parameters that define the characteristics of the axis while operating in simulation mode. The following simulation parameters are provided:
  - *NegativeLimitSwitch* (float: encoder counts) and PositiveLimitSwitch (float: encoder counts) – The simulated position of the negative and positive limit switches. These positions are used to set the limit flags when the motor position exceeds these limits. They are also used to generate a random initial starting position for the axis.
  - *LimitsEnabled* (Boolean) – This field specifies whether the limits are enabled or not. Disabling limits is useful for motors that can rotate indefinitely without reaching a limit, such as a filter wheel.
  - *MaximumVelocity* (float: *Units* per second) and *MaximumAcceleration* (float: *Units* per second per second) – Specifies the maximum speed and acceleration of the simulated motor. These are used to limit jogging and tracking moves.
  - *HomingRate* (float) – This field determines how fast the simulated motor homes. It is a factor of the MaximumVelocity. A setting larger than 1 will decrease the time it takes to home the simulated axis.
  - *PositionJitter* (float: encoder counts) – Specifies the peak value of a uniform distribution of random noise added to the motor position in order to simulate encoder noise on the axis.
  - *SpeedJitter* (float: encoder counts per *Unit* per second) – Similar to PositionJitter, this parameter is multiplied by the current commanded speed of the motor to simulate an increase in jitter magnitude for faster motion.
  - *JitterRate* (float: encoder counts per second) – Specifies the rate of change of simulated jitter, effectively this determines how quickly the motor jitters.

4.2.2.2 Schema and Operation

The operation of the TelescopeServer as a citizen on the larger Framework is determined by the Schema-TelescopeServer.dms file. The schema file defines attributes and commands used to manipulate the telescope via the Framework. Attributes are published structures containing status information and can be retrieved by clients either by subscription or by using a status retrieving command.

Commands and attributes are sent and received using the EOS Control System Client API. The Device Browser application allows access to this interface without needing to write a client application, however operators should rarely need to manipulate servers using the schema directly. Instead, they should use high-level user interfaces, such as the TelescopeInterface, which may present the attributes and commands in manner that does not resemble the interface described here. Operators should refer to the documentation of the client application.

4.2.2.2.1 ApplicationInfo

The *ApplicationInfoStruct* is a standardized attribute structure that is used by most EOST servers. It contains information about the TelescopeServer as it is running. The *ApplicationInfoStruct* has the following fields:

- *Version* (string) - A string containing the version number of the TelescopeServer. The current TelescopeServer will report "3.3.4.15" if using the 2.1.2 version of the EOS Framework.
- *BuildInfo* (string) - A string containing the date and time that the TelescopeServer was built. The date string will resemble: "Jun 30 2009 14:57:34."
- *SchemaVersionMajor* (integer) - A number representing the major schema version of the server. This will correspond with the X in the *__Schema_Version_Major__X* field of the *__Schema_Version__* structure if the server was built using the same schema as it is currently running with.
- *SchemaVersionMinor* (integer) - A number representing the minor schema version of the server. This will correspond with the Y in the *__Schema_Version_Minor__Y* field of the *__Schema_Version__* structure if the server was built using the same schema as it is currently running with.
- *Simulated* (boolean) - A flag that indicates if the server is running in 'Simulate' mode. The flag will be false if the server is operating normally.
- *Trace* (boolean) - A flag that indicates if the server is running with additional tracing turned on. If true, additional debugging information is being logged in the server's log file.
- *Fault* (*EventInfo*) - A structure that indicates why the server may currently be in a fault mode. The server will transition to 'Error' mode when a fault requiring operator intervention has occurred. The fault field will report information about the error that has occurred, however additional investigation, including examining log files may be required to determine the proper course of action. Sending the server the software Reset command will clear the fault field. The *EventInfo* structure contains the following fields:
    - Reason (string) - A string representing the likely cause of the error state.
    - Date (string) - A string representing the date and time the fault occurred.

4.2.2.2.2 Commands

The TelescopeServer will respond to several common top-level commands that are common to most EOST servers. All command structures of servers written by EOST will contain two sub-structures: Parameters and Returns. The Parameters structure indicates the command parameters that may be required by the server. The Returns structure indicates the structure of the data that will be returned by the server upon successful operation. The following top level commands are for manipulating and retrieving information about the server independently from the logical device:

- *GetApplicationInfo* - Returns an *ApplicationInfo* structure, which matches the structure published in the Attributes section. This command provides an alternate method of retrieving the structure that does not require a subscription.
- *Reset* - If the server is in an Error state, this command will attempt to reset the server and clear the error state.
- *DebugTrace* - This debug command allows an operator to enable or disable the trace mode of the server while it is running. This turns on additional logging that may be useful for debugging purposes. **Note:** Debug commands may be removed on production installations.
- *DebugSetFault* - This debug command artificially instigates a server fault. This may be useful for debugging purposes. **Note:** Debug commands may be removed on production installations.

4.2.2.2.3 Devices

TelescopeServer uses logical devices to group related functions. Each device has a structure indicating published attributes and a structure indicating the commands that can be used to operate the device.

The primary means for controlling and monitoring the telescope motion overall is the *Telescope* device. It defines the following Attributes:

- *Status* (TelescopeStatusStruct) – This field gives most of the current information about the telescope, with the notable exception of pointing which is in a separate field. The struct includes the following fields:
    - *Recalculating* (boolean) – This field indicates whether the telescope server is computing a new track segment. If so, the other fields in this structure may not be self-consistent. The fields are picked off whenever a Status request is made rather than waiting for a commensurate snapshot but this means the fields may be in a state of change when the snapshot is made.
    - *State* (string) – This is the current overall state of the telescope. The possible states are defined as follows:
        - “Disabled” — The motors are off and the telescope is slack. The telescope starts in disabled mode when the telescope control system is started, and may be placed in disabled mode at any later time by sending the Disable command. The telescope also enters disabled mode if it encounters certain types of tracking errors, such as triggering Sun avoidance.
        - “Homing” — One or more telescope axes are seeking an absolute position reference so that the encoder counts reported from their incremental encoders represent a true position of each axis. If necessary, the telescope will start homing automatically before tracking an object or

moving to the park position (if the AutoHome flag is set in the SetTrack or Park commands). The telescope may be explicitly commanded by a client to start homing using Home or HomeAll commands.

- "Slewing" — The telescope is currently moving at maximum speed to reach the target position of the current track loaded by a client using the SetTrack command. Some of the telescope axes may already be in position, but at least one is still moving into position. The telescope may also switch to this mode if the client adjusts the secondary focus, the optional instrument rotator, the tertiary mirror, or selects a different instrument port.

- "Tracking" — The telescope is currently following the current track loaded by the SetTrack command. All of the telescope axes are in position.

- "Parking" — The telescope is currently moving at maximum speed to reach the park position for each axis. A client commands the telescope to move to its park position by sending the Park command. Some of the telescope axes may already be in their park position, but at least one is still moving into position.

- "Parked" — The telescope has reached its park position and is currently stationary at this position under servo control.

- "Stopping" — The telescope axes are currently slowing down to stop at a position other than the telescope park position. A client commands the telescope to stop by sending the Stop command. The telescope may also stop if the client loses or releases control during tracking or parking. Some of the telescope axes may already be stopped, but at least one is still moving.

- "Stopped" — All telescope axes are stationary and holding position under servo control, but the telescope is not in its park position. A client commands the telescope to stop by sending the Stop command. The telescope may also stop if the client loses or releases control during tracking or parking. This state should not be confused with tracking a stationary target; in this case the Telescope Mode field will be set to "Tracking".

- *HomedOkay* (boolean) – This field is set to true if the last homing operation completed successfully. The telescope must be homed successfully before it can be used to track a target or moved to its park position. This field is set to false if the last homing operation was not successful (the HomingFailed field will be true), if homing is currently in progress (the HomingFailed field will be false and the Mode field will be set to "Homing"), or if the telescope has not been homed since it was started (the Telescope Homing Failed field will be false and the Mode field will be set to "Disabled").

- *HomingFailed* (Boolean) – This field is set to true if the last homing operation failed to complete successfully. This field is set to false if the last homing operation was successful (the HomedOkay field will be true), if homing is currently in progress (the HomedOkay field will be false and the Telescope Mode field will be set to “Homing”), or if the telescope has not been homed since it was started (the HomedOkay field will be false and the Telescope Mode field will be set to “Disabled”).

- *Target* (string) – This is a string containing the name of the current target to be or currently being tracked. The string has no real meaning and may be left empty.

- *TimeToLimit* (float: seconds) – This is an estimate of the time remaining under control of the current command until any axis reaches its software limit.

- *MountModelLimit* (boolean) – This field will be set to True if the position offset due to the mount model for the current telescope position has reached a maximum positive or negative value for either the azimuth or elevation axis. The telescope can be operated safely if this flag is set to True, but the pointing accuracy of the telescope may be degraded at this position.

- *PositionLimit* (Boolean) – This field is set to true if the telescope is given a command while tracking that would have moved any of the telescope axes past their normal range of motion. This may occur if the currently loaded track exceeds an axis position limit after some time (for example, the telescope reaches an azimuth limit after tracking a constant RA/Dec position for several hours), or if a client issues a command such as SetOffset that moves the telescope into an axis position limit (typically this can only happen if the telescope is already operating very close to the axis position limit). If a position limit is reached, the telescope will stop tracking and this field will be set to true. This field is set to false if the telescope is operating in its normal range of motion, or if the telescope has not been homed

- *TimeToSkyline* (float: seconds) – This indicates the predicted time remaining under the control of the current command until any axis reaches the software defined horizon. *This field is currently not implemented and will always return zero.*

- *SkylineLimit* (Boolean) – This field is set to true if the telescope is given a command while tracking that would have moved the telescope below the allowable skyline limit. The skyline limit is a minimum allowable elevation angle for each azimuth angle of the telescope, intended to prevent the telescope from colliding with fixed obstacles in the dome. This field may be set to true if the currently loaded track exceeds the skyline limit after some time (for example, the telescope reaches the skyline limit after tracking a constant RA/Dec position for several hours), or if a client issues a command such as SetOffset that moves the telescope position into the skyline limit (typically this can only happen if the telescope is already operating very close to the skyline limit). If the skyline limit is reached, the telescope will stop tracking and this field will be set to true. This field is set to false if the telescope is operating above the skyline limit, or if the telescope is not currently tracking. The skyline limit is defined in the configuration file TelescopeServer.dmx.

- *SunAvoidanceEnabled* (Boolean) – This indicates whether the sun avoidance capability is currently enabled.

- *SunAvoidance* (Boolean) – This field is set to true if the Sun avoidance function is enabled and the telescope is pointing too close to the Sun. If this happens, the telescope will be disabled. This field is set to false if the telescope is not considered to be pointing too close to the Sun, or the Sun avoidance function is disabled. Both the telescope position and the calculated position of the Sun are used to determine whether the telescope is pointing too close to the Sun. If the telescope position is not currently known (that is, it has not been homed successfully) then only the Sun position is considered by checking to see whether it is above a certain elevation angle, known as the solar horizon. If there is a problem with the telescope clock or the telescope site coordinates, then the calculated Sun position is considered to be unreliable, and the telescope is considered to be at risk regardless of where it is pointing. Parameters controlling Sun avoidance are defined in a configuration file TelescopeServer.dmx.

- *ClockError* (Boolean) – This field is set to true if a problem with the telescope clock is detected. The telescope clock is checked periodically against the time reported by the GPS receiver, as well as the clock on the telescope control computer. If a discrepancy is found, this field will be set to true. A problem may also occur if the telescope clock can not be reset. In this case, the telescope will be disabled.

- *ServoError* (Boolean) – This indicates whether any of the telescope servos is currently in an error state. Servo errors can be caused by several reasons including excessive following error or amplifier fault.

- *InstrumentPort* (string) – This names the currently selected instrument port. By association, this also represents a certain configuration for the tertiary mirror, optional instrument rotator, focus offset and mount model.

- *MountModel* (string) – This names the currently active mount model.

- *Rotator* (string) – This field is not used on the Lick APF.

- *Axes* (array of AxisStatus) – This array provides details of the individual axes of the telescope. It is comprised of the following fields (which are in direct correspondence to their fields in the general telescope status):

    - *Name* (string) – This is a descriptive name of the axis.

- *State* (string) – This is the current state of this axis. The states include: Disabled, Homing, Slewing, Tracking, Parking, Parked and Stopped.
- *HomedOkay* (boolean) - This indicates whether this axis has been successfully homed.
- *HomingFailed* (boolean) - This indicates whether this axis has attempted to home but failed to do so.
- *TimeToLimit* (float: seconds) - This is an estimate of the time remaining under control of the current command until this axis reaches its software limit
- *PositionLimit* (boolean) - This indicates whether this axis has been commanded to a position that is outside its normal range of motion
- *SpeedLimit* (boolean) – This indicates whether the current command is asking for this axis to move faster than the maximum safe limit. If true, the speed limit is being enforced and the axis is not following the commanded trajectory.
- *PositionFault* (boolean) – This indicates this axis has encountered an end-of-travel limit. If this occurs the axis must be re-homed.
- *FollowingErrorWarning* (boolean) – This indicates whether the difference between the commanded and actual encoder position for this axis is currently above a low water mark. This is not an error, but indicates the axis is nearing its maximum allowed tracking error.
- *FollowingErrorFatal* (boolean) – This indicates whether the difference between the commanded and actual encoder position has exceeded the maximum allowed. This may indicate the axis has encountered an obstacle or something about the motor or encoder electronics has failed.
- *AmplifierFault* (boolean) – This indicates the amplifier running the motor for this axis is reporting an error. It may indicate the amplifier has over heated due to excessive load; has detected a short circuit; is being asked to supply more than the maximum allowed sustained or peak current; or the supply voltage is incorrect.
- *EncoderError* (boolean) – This indicates the position encoder for this axis is reporting an error. It may indicate the encoder is not connected; is not reporting motion; or has lost power.
- *OtherFault* (boolean) – This indicates this axis has encountered an error other than those reported above.

  - *FocusPositionLimit* (FocusLimitStruct) – Fields in this struct indicate whether each of the possible secondary motion limits have been encountered. If so, try re-homing the secondary.

- *Position* (TelescopeFullPositionStruct) – This is a composite structure containing the telescope position and motion in compete detail. It consists of the following sub-structures:

  - *Time* (TelescopeTimeStruct) – This contains the current UTC and Sidereal times as they are known to the TelescopeServer (If you just want to know the time, read it from the GPSServer in§4.2.3).

  - *CmdPos* (TelescopePositionStruct) –
  - *ActPos* (TelescopePositionStruct) –
  - *CmdVel* (TelescopePositionStruct) –
  - *ActVel* (TelescopePositionStruct) – These fields all use the same struct to report the most commonly used information of current and actual positions and velocities of the telescope. In the case of velocities, each field reports the rate-of-change per second of the stated quantity. The *commanded* information will be set to and remain equal to the commanded position of a new move. The *actual* information will reflect the current position at the moment of the report. This struct has the following fields:

    - *Az* (float: radians) – This is the azimuth of the telescope, measured clockwise (through East) from celestial North.

    - *El* (float: radians) – This is the elevation of the telescope, measured up from the plane defining the local horizon. This is purely geometric without regard to refraction, *i.e.,* as if the telescope were situated "outside" the atmosphere.

    - *RAJ2000* (float: radians) – This is the astrometric Right Ascencion in J2000 coordinates towards which the telescope is now pointing.

    - *DecJ2000* (float: radians) – This is the astrometric Declination in J2000 coordinates towards which the telescope is now pointing.

    - *RAApparent* (float: radians) – This is the apparent Right Ascension coordinate at the epoch of date towards which the telescope is now pointing.

    - *DecApparent* (float: radians) – This is the apparent Declination coordinate at the epoch of date towards which the telescope is now pointing.

    - *HA* (float: radians) – This is the hour angle meridian towards which the telescope is now pointing. It is numerically equal to the difference between the current sidereal time and the apparent RA. The value can be entered in any circular system and it will be converted internally to the range 0 .. 2*π.

  - *PA* (float: radians) – This is the parallactic angle of the position towards which the telescope is currently pointing. It is the angle subtended on the celestial sphere between the great circle arc through the telescope position and the zenith and the arc through the telescope position and the North Celestial Pole.

  - *Airmass* (float: unitless) – This is the number of canonical atmospheres through which a ray leaving the telescope traverses until it reaches the vacuum of space. It is defined as 1 at the zenith and increases approximately as the secant of the zenith angle.

- *DetPos* (TelescopeDetailedPositionStruct) –
- *DetVel* (TelescopeDetailedPositionStruct) - These fields both use the same struct to report additional rather more esoteric information about telescope position and motion, as well as Solar positions. In the case of velocities, each field reports the rate-of-change per second of the stated quantity.

- *Focus* (FocusFullPositionStruct) – This field contains details of the focus system. The motion coordinates are defined in §4.2.2.1.6. It includes the following fields:

  - *Enabled* (boolean) – This indicates whether the focus system is currently enabled at all.

  - *Type* (string) – This field indicates whether the focus system is a Hexapod or a Quasi-static mount.

  - *CmdAbsPos* (FocusPositionStruct) – This struct contains the currently commanded values for focus, tip, tilt, and decentering x and y.

  - *CmdCompPos* (FocusPosiionStruct) – This struct contains the currently commanded values for focus, tip, tilt and decentering x and y after the various compensation tables for elevation, temperature and range have been applied.

  - *ActAbsPos* (FocusPositionStruct) – This struct contains the current values for focus, tip, tilt, and decentering x and y.

  - *ActCompPos* (FocusPosiionStruct) – This struct contains the current values for focus, tip, tilt and decentering x and y after the various compensation tables for elevation, temperature and range have been applied.

- *Rotator* (RotatorPositionStruct) – This field is not used in this telescope.

- *Offset* (OffsetStruct) – This structure captures the current net pointing offsets being injected into the nominal telescope pointing direction in each of the four supported offset coordinate system. All offsets are in radians. The OffsetStruct contains the following fields:
    - *Az* (float) – the current offset in azimuth, on-sky.
    - *El* (float) – the current offset in elevation.
    - *RA* (float) – the current offset in Right Ascension.
    - *Dec* (float) – the current offset in Declination.
    - *InstrumentX* (float) – current offset in the X instrument direction.
    - *InstrumentY* (float) – current offset in the Y instrument direction.
    - *Along* (float) – current offset tangent to the direction of the track being followed. Positive is in the direction of a traveler following the track forward in time.
    - *Across* (float) – current offset perpendicular to the direction of the track being followed. Positive is to the right as seen by a traveler moving along the track forward in time.
- *Mets* (MetsStruct) – This structure captures the current values of atmospheric temperature (°C), pressure (Pascals) and humidity (percent relative) being used to account for refraction. The values should be set by using the SetMets message once before using the telescope and then whenever they change significantly. The temperature and humidity sensors built into the telescope are *not* used to define these values because it is presumed they are better to come from outdoor equipment.
- *TrussTemp* (TrussTempStruct) – This generally contains the arithmetic mean of the several temperature sensors attached to the telescope truss. The value is in °C. This value is used to compute the focus compensation due to change in truss length as a function of temperature. Note the Telescope Server itself is completely unaware of the temperature sensors on the truss. It depends on some other process to transmit their average value occasionally via the SetTrussTemp command.
- *Refraction* (RefractionStruct) – This contains the target distance and wavelength of light for which refraction is currently being computed. The distance and wavelength are both in meters. The wavelength value must be between 1 and 5000 nanometers. The distance can be zero to specific infinity. The values should be set by using the SetRefraction message once before using the telescope and then whenever a different value is desired.

All command structures of servers written by EOST will contain two sub-structures: Parameters and Returns. The Parameters structure indicates the command parameters that may be required by the server. The Returns structure indicates the structure of the data that will be returned by the server upon successful operation.

The *Telescope* device defines the following Commands:

- *GetStatus* returns a *TelescopeStatusStruct*, the same as for the *Status* Attribute, see § 4.2.2.2.3. The only parameter, *AxisData*, is a Boolean to select whether to not to include the Axis array of AxisStatus in the return.

- *GetSiteInfo* returns a *SiteInfoStruct* which contains the following fields:

  - *Latitude* (float) – geodetic latitude of the telescope location, in radians. This is the angle a ray through the site to the zenith makes with the equatorial plane. It is normal to the standard ellipsoid at the site location and in general does not go through the center of the Earth. Also known as geographic latitude, it is the quantity generally shown on maps.

  - *Longitude* (float) – longitude of the telescope location, in radians with positive defined as east of the prime meridian.

  - *Height* (float) – height of the telescope location in meters above mean sea level.

- *GetAxisInfo* returns information about the performance capabilities of each telescope axis. It consists of a struct that in turn contains the following three fields:

  - *Axes* (array of AxisInfoStruct). This is an array of struct for each telescope axis with the following fields:

    - *Name* (string) – short descriptive name of axis
    - *Units* (string) – This field identifies the real-world base units that will be used for specifying several of the position parameters noted below. Valid units may be specified from the following list:
      - Radians
      - Degrees
      - Arcminutes
      - Arcseconds
      - Microradians
      - Hours
      - Revolutions
      - Meters
      - Metres
      - Millimeters
      - Millimetres
      - Microns

        - EncoderCounts
        - Encoder Counts
      - *MinPos* (float) – minimum axis range, in encoder counts.
      - *MaxPos* (float) – maximum axis range, in encoder counts
      - *MaxSpeed* (float) – maximum axis speed, in encoder counts per second.
      - *MaxAccel* (float) – maximum axis acceleration, in encoder counts per sec^2.
      - *ParkPos* (float) – park position
      - *Scale* (float) – encoder counts per Unit.
      - *Offset* (float) – encoder counts offset from an absolute reference
    - *FocusMinPos* and *FocusMaxPos* (FocusPositionStruct) – Although focus may not strictly be implemented as one control axis, it is generally useful when considering operational capability of the telescope so this command returns information for minimum and maximum focus positioning using the fields of these two structs. This struct contains the following fields (see 4.2.2.1.6 for definitions of terms):
      - *Focus* (float) – focus position, meters.
      - *Tip* (float) – elevation, on-sky radians
      - *Tilt* (float) – cross-elevation, on-sky radians
      - *DecenterX* (float) – x-axis offset, meters
      - *DecenterY* (float) - y-axis offset, meters.
- *GetAxisPosition* returns actual and commanded position information about each axis in terms of low-level encoder positions that are each commensurate at one moment of time. It consists of the follow fields:
    - *Time* (struct TelescopeTimeStruct) contains the real and sidereal time at which the axis positions are valid. The struct contains the following fields:
      - *Time* (PrecisionDateStruct) – the telescope time at which the axis positions are being reported. Telescope time is the moment at which the current track loaded into the PMAC began.
      - *Sidereal* (float) – the sidereal time at which the axis positions are bring reported.

        - *TimeReal* (PrecisionDateStruct) – the UTC at which the axis positions are being reported.

    - *Data* (array of AxisPositionStruct) contains the actual and commanded positions for each axis. The struct contains the following fields:

        - *Name* (string) – short descriptive name of axis

        - *Cmd* (float) – commanded position of axis, encoder counts.

        - *Act* (float) – actual position of axis, encoder counts.

- *GetPosition* returns actual and commanded position information about each axis in astronomical terms that are each commensurate at one moment of time. The return consists on the following field:

    - *Data* (struct TelescopeFullPositionStruct) – info about each axis and when the info was taken. For a description of the fields in this struct, refer to the Position Attribute in §4.2.2.2.3.

- *Reset:* If the Telescope Server is in an Error state, this command will attempt to reset the server and clear all faults to return the Server state to Ready. The command will fail if the faults can not be cleared. This command will not interrupt any operations currently in progress such as tracking or slewing.

- *HomeAll* commands that each axis commence homing. All axes will be homed simultaneously unless specified otherwise with the parameter MustHomeAfter in the ServoAxisConfig TelescopeServer.dmx configuration settings. The response to this command is nearly immediate and indicates only whether the command was successfully begun, not that the home procedures themselves have been accomplished. Homing is not allowed if the telescope has not yet been initialized or if it is currently tracking.

- *Home* commands each specified axis to commence homing. All named axes will be homed simultaneously unless specified otherwise with the parameter MustHomeAfter in the ServoAxisConfig TelescopeServer.dmx configuration settings. The response to this command is nearly immediate and indicates only whether the command was successfully begun, not that the home procedure has been accomplished. Homing is not allowed if the telescope has not yet been initialized or if it is currently tracking.

- *GetTrack* returns the current track and *SetTrack* defines the next track. A track is a table of times and positions the telescope is to follow. A track is defined with a *TrackStruct.*which has the following fields:

    - *Target* (string) – a name for the target being tracked. This is only recorded for convenience it is not used by the system for anything meaningful.

- *TrackType* (string) – Selects one of three different spherical coordinate systems to use for specifying track positions. The accepted string possibilities are “Az/El”, “HA/Dec” or “RA/Dec”vis Tele. This establishes the coordinate system for the entries in the next field:

- *Points* (array of TrackPointStruct) – array of points defining the track. The *TrackPointStruct* contains the following fields:

  - *Time* (float) – time relative to start of track, in seconds. The Time values in successive array entries must be in strictly increasing order.

  - *Pos1* (float) – position of this point in the first spherical coordinate as specified in the *TrackType* parameter, *i.e.*, this will be azimuth, HA or RA, always in radians.

  - *Pos2* (float) – position of this point in the second coordinate as specified in the *TrackType* parameter, *i.e.*, this will be elevation or Declination, always in radians.

  The remaining fields in the *TrackPointStruct* are optional. They must either all be absent or all be specified for all points in the track. As with *Pos1* and *Pos2*, they come in pairs which refer to the first and second spherical coordinate as specified in the *TrackType* parameter.

  - *Vel1* and *Vel2* (float) – these specify the required velocity of this point, in radians/sec.

  - *Acc1* and *Acc2* (float) – these specify the required acceleration of this point, in radians/sec^2.

  - *Jerk1* and *Jerk2* (float) – these specify the required jerk of this point, in radians/sec^3.

- *StartTime* (PrecisionDateStruct) – Specifies the UTC date and time when this track is to be started. This field is optional. If it is not specified, the track will start immediately. If the start time specified lies in the future, the telescope will move to the first point in the track and wait. If the start time specified has already passed then the track will be picked up at its current position. If the time for the entire track has already passed then the telescope will move to the last point in the track and stop. *PrecisionDateStruct* consists of the following fields:

  - *Year* (integer) – year, A.D. four digits.

  - *Month* (integer) – month in year, 1 .. 12

  - *Day* (integer) – day in month, 1 .. 31

    - *DayOfYear* (integer) – day in year, 1 .. 366. Note: either this field or Month and Day should be used, not both.
    - *Hour* (integer) – hour in day, 0 .. 23
    - *Minute* (integer) – minute in hour, 0 .. 59
    - *Second* (integer) – second in minute, 0 .. 59.
    - *MicroSecond* (integer) – microsecond in second, 0 .. 999999
    - *PicoSecond* (integer) – picosecond in microsecond, 0 .. 999999
    - *AttoSecond* (integer) – attosecond in picosecond, 0 .. 999999

- *AutoHome* (Boolean) – whether to first home the telescope if not already homed before performing the track.
- *ProperMotionRA* (float) – specifies the component of apparent rate of motion of the target projected along a meridian of J2000 Right Ascension, in radians/sec. This motion is added to the track specified in the Points array based on the elapsed time from StartTime.
- *ProperMotionDec* (float) – specifies the component of apparent rate of motion of the target projected along a latitude of J2000 Declination, in radians/sec. This motion is added to the track specified in the Points array based on the elapsed time from StartTime.
- *Parallax* (float) – specifies the apparent angular displacement of the target as seen from the Earth and the Sun, in radians. This in turn is used to compute a distance to the target for projecting each track position to equatorial coordinates.
- *NormalizeAzEl* (Boolean) – specifies whether to use the reference positions specified in the fields ReferencePosAz and ReferencePosEl. If this field is False, those two fields are ignored.
- *ReferencePosAz* (float) – specifies an absolute azimuth position within the full range of the telescope motion. If NormalizeAzEl is True, then when converting the canonical track azimuth positions to absolute azimuth positions for the telescope, the initial azimuth position will be chosen so as to be within 180° of this value. The purpose of this is to allow the commanding application to avoid the possibility of encountering an end-of-travel limit during the course of a long track. If NormalizeAzEl is False, then the track will begin at whatever azimuth happens to be nearest to the current telescope azimuth.
- *ReferencePosEl* (float) – specifies an absolute elevation position within the full range of the telescope motion. If NormalizeAzEl is True, then when converting the canonical track azimuth and elevation positions to absolute positions for the

telescope, the telescope will be forced into dump mode if this value is greater than 90°. If NormalizeAzEl is False, the telescope will not use dump mode. Attempts to use dump mode will be silently ignored if the telescope is not physically capable of elevations above 90°.

- *SetSlew* commands the telescope to move to and hold a specified celestial position. This command is essentially the same as loading a single point track. It is especially useful when using the Device Browser which does not have the ability to enter the array necessary to perform a SetTrack. The response to this command is nearly immediate and indicates only whether the command was successfully begun, not that the slew has been accomplished. This command uses the SlewStruct which has the following fields:

    - *Target* (string) – a name for the target position. This is only recorded for convenience it is not used by the system for anything meaningful.

    - *TrackType* (string) – Selects one of three different spherical coordinate systems to use for specifying a slew position. The accepted string possibilities are “Az/El”, “HA/Dec” and “RA/Dec”. This establishes the coordinate system for the entries in the next two fields:

    - *Pos1* (float) – position of the slew in the first spherical coordinate as specified in the *TrackType* parameter, *i.e.*, this will be azimuth, HA or RA, always in radians.

    - *Pos2* (float) – position of the slew in the second coordinate as specified in the *TrackType* parameter, *i.e.*, this will be elevation or Declination, always in radians.

    - *AutoHome* (Boolean) – whether to first home the telescope if not already homed before performing the slew.

    - *ProperMotionRA* (float) – specifies the component of apparent rate of motion of the target projected along a meridian of J2000 Right Ascension, in radians/sec. This motion is added to the target position specified above based on the elapsed time from StartTime.

    - *ProperMotionDec* (float) – specifies the component of apparent rate of motion of the target projected along a latitude of J2000 Declination, in radians/sec. This motion is added to the target position specified above based on the elapsed time from StartTime.

    - *Parallax* (float) – specifies the apparent angular displacement of the target as seen from the Earth and the Sun, in radians. This in turn is used to compute a distance to the target for projecting each track position to equatorial coordinates.

    - *NormalizeAzEl* (Boolean) – specifies whether to use the reference positions specified in the fields ReferencePosAz and ReferencePosEl. If this field is False, these two fields are ignored.

  - *ReferencePosAz* (float) – specifies an absolute azimuth position within the full range of the telescope motion. If NormalizeAzEl is True, then when converting the canonical slew azimuth position to absolute azimuth position for the telescope, the azimuth position will be chosen so as to be within 180° of this value. The purpose of this is just to be analogous to the same capability of the SetTrack command. If NormalizeAzEl is False, then the slew will be performed at whatever azimuth happens to be nearest to the current telescope azimuth.

  - *ReferencePosEl* (float) – specifies an absolute elevation position within the full range of the telescope motion. If NormalizeAzEl is True, then when converting the canonical slew azimuth and elevation positions to absolute positions for the telescope, the telescope will be forced into dump mode if this value is greater than 90°. If NormalizeAzEl is False, the telescope will not use dump mode. Attempts to use dump mode will be silently ignored if the telescope is not physically capable of elevations above 90°.

- *Park* commands the telescope to slew each axis to its ParkPosition as specified in the ServoAxisConfig entry of each axis in the TelescopeServer.dmx configuration file. The response to this command is nearly immediate and indicates only whether the command was successfully begun, not that the park has been accomplished. The Park command uses PartStruct which has the following field:

  - *AutoHome* (Boolean) – whether to first automatically each axis if not already homed before performing the park operation.

- *Stop* commands the telescope to bring each axis to a controlled stop and hold position. If the telescope has successfully been Homed, this command is similar in effect to loading a single-point track for the stop position. The response to this command is nearly immediate and indicates only whether the command was successfully begun, not that the stop has been accomplished. Commanding Stop cancels any existing track, homing or park operation that might be underway. If the telescope is already stopped or disabled, this command reports to succeed but has no real effect. If currently tracking, it does not remove the track from memory so it may still be retrieved using GetTrack if desired. After the Stop command is issued, any commands that set individual axes, such as SetFocus, will record the new settings but will not take effect until the SetTrack command is issued. If the telescope has not yet been successfully Homed, this command acts like a Disable command.

- *Disable* commands the telescope to immediately shut off the motors to each axis and allow them to go slack. The axis may not stop immediately depending on imbalance, inertia and other ancillary effects. After the Disable command is issued, any commands that set individual axes, such as SetFocus, will record the new settings but will not take effect until the SetTrack command is issued.

- *GetInstrumentPortList* returns an array of InstrumentPortStruct, one for each port. The InstrumentPortStruct contains the following fields:

   - *Name* (string) - short descriptive name of this port.
   - *Orientation* (float) – angle to be considered zero degrees when commanding and reporting the rotation angle of this instrument port, specified as an angle clockwise from zenith, in radians.
   - *MirrorImage* (Boolean) – indicates whether the image presented by the instrument at this point is flipped. This is used to effectively change the sign of the angles used when commanding and reporting rotation angle of this port.
   - *FocusOffset* (float) – offset added to all commanded and reported focus positions for this port, in meters.
   - *PortSelectors* (array of PortSelectorStruct) – lists the required position of each axis involved in utilizing this instrument port. The positions are indicated in a list of PortSelectror structs which have the following fields:
      - *ServoAxis* (string) – the standard name of the axis
      - *Position* (float) – the position of this axis to select the given port, in encoder counts.
   - *Rotator* (string) – name of the rotator axis associated with this instrument port, if any.
   - *MountModel* (string) – name of the model to be employed when utilizing this instrument port. Uses the default model if not specified.

- *SelectInstrumentPort* commands the telescope to move all relevant axes involved in port selection to set up for the specified instrument position. The command requires one argument: a string *Name* that matches the name of a previously configured instrument port; case is not significant.
- *GetInstrumentPort* returns an InstrumentPortStruct that contains information about the currently selected port. See the commnd GetInstrumentPortList for a description of the fields within an InstrumentPortStruct.
- *GetOffset* returns an OffsetStruct describing any pointing offsets currently in effect. For a description of the individual fields, see the Offset Attribute in §4.2.2.2.3.
- *SetOffset* commands the telescope to point away by small amounts from the current track trajectory. The net offsets are specified in an OffsetStruct. For a description of the individual fields, see the Offset Attribute in §4.2.2.2.3. There is no mechanism to incrementally add to the current offset. The offsets are in four sets of naturally orthogonal pairs. More than one pair of offsets may be set simultaneously but multiple offsets larger than a degree or so may interact and deviate slightly from the expected position. Thus, it is recommended that offsets be applied one pair at a time. If an offset is

set that would cause the telescope to exceed a position limit, tracking is stopped and the PositionLimit field from GetStatus is set to True. Commanded position offsets are independent of any adjustments to telescope position due to the mount model, secondary mirror position, tertiary mirror position or atmospheric refraction.

- *SetFocus* commands a new focus position, including tip, tilt and decentering. The command uses the FocusPositionStruct. For a description of the fields in this struct see §4.2.2.1.6.

- *SetTertiary* commands a new position for the tertiary mirror. It uses the TertiaryOffsetStruct which contains the following fields:

    - *Tip* (float) - The angle tilting around a line through the tertiary mirror perpendicular to the main telescope optical axis, and perpendicular to the optical axis from the instrument port to which the tertiary mirror is directed, such that a positive value represents the top of the tertiary mirror moving towards the instrument port, in radians.

    - *Tilt* (float) - The angle tilting around a line through the tertiary mirror parallel to the main telescope optical axis, such that a positive value represents the tertiary mirror rotating clockwise as viewed from the secondary mirror looking towards the primary mirror, in radians.

- *SetRotator* commands a new position for the rotator. It uses the RotatorStruct which contains the following fields:

    - *Mode* (string) – This field specifies the mode of operation for the instrument rotator. The instrument rotator can operate in any of three modes. The valid values for this field are (the strings are case-insensitive):

        - "Stationary" - The rotator is held in a fixed position with respect to its home (zero) position. The Orientation value determines what this position is: zero aligns the top of the instrument at the 12 o'clock position, and positive is in the clockwise direction when looking at the rotator from the instrument side if the telescope is pointing at the North horizon.

        - "Zenith" — The rotator is turned to maintain the instrument image at a constant orientation with respect to zenith. The Orientation value determines what this orientation is: zero aligns the top of the instrument image with zenith, and positive is in the clockwise direction (from zenith towards the East if the telescope is pointing at the North horizon).

        - "Celestial" — The rotator is turned to maintain the instrument image at a constant orientation with respect to the celestial sphere. The Orientation value determines what this orientation is: zero aligns the top of the instrument image with the North Celestial pole, and positive is in the direction clockwise on the sky from Celestial North.

- *MaxRun* (Boolean) – If True, the rotator will automatically be positioned to the lap which corresponds to the given Orientation closest to the middle of the physical rotational range of the rotator mechanism. The intention is to set up for maximum uninterrupted tracking time before encountering a rotator limit. If this value is False, the Orientation is simply used directly.

- *Orientation* (float) – This fields specifies the commanded orientation of the instrument to be maintained by the instrument rotator. The reference frame for the orientation depends on the Mode field (above). The angle is in radians. Any value is valid, and will be normalized between 0 and 360.

- *SetTimeBias* sets a time interval to be added to the flow of actual time at which the telescope is to function. The intent of this bias is to aid tracking earth satellites, when the error in a potentially aged ephermeris is predominantly in terms of time along the track. The interval is specified using a PrecisionIntervalStruct which has the following fields:
  - *Days* (integer) – number of whole days
  - *Seconds* (integer) – number of seconds in days, 0 .. 86,399
  - *NanoSeconds* (integer) – number of nanoseconds in seconds, 0 .. 999,999,999
  - *AttoSeconds* (integer) – number of attoseconds in nanosecond, 0 .. 999,999,999

- *SetMets* informs the telescope of meteorological values to be used to compute refraction. It uses the MetsStruct which has the following fields:
  - *Temperature* (float) – atmospheric temperature, °C
  - *Pressure* (float) – atmospheric pressure, Pascals (1 Pascal == 0.01 mBar)
  - *Humidity* (float) – atmospheric humidity, percent relative

- *SetTrussTemp* sets the temperature to be used as the input to the focus compensation algorithm. Note the Telescope Server itself is completely unaware of the temperature sensors on the truss. It depends on some other process to transmit their average value occasionally via this command. This command uses the TrussTemperatureStruct which contains the following field:
  - *Temperature* (float) - the temperature to be used for focus compensation, in °C.

- *SetRefraction* sets the values to be used for refraction other than the meteorological contributions. This command uses the RefractionStruct which contains the following fields:
  - *Wavelength* (float) – wavelength of light, meters

  - *TargetDistance* (float) – distance to target, meters; use 0 for infinite

- *SelectMountModel* specifies the name of the desired mount model to use. It takes a single string argument containing the name of an existing model. The name comparison is not case sensitive.

- *SetMountModel* installs and saves a mount model to be used for subsequent pointing maneuvers. Mount models are saved to the file TelescopeMountModel.dmx in the folder C:\eos\bin\servers\ConfigServer\SettableData. This command uses the MountModelStruct which contains the following fields:

  - *Name* (string) – a short descriptive name for this model. If this name matches an existing model, the values are updated. If this name is blank, it automatically matches the currently loaded model and its values are updated. If this name does not match any existing model, a new model is created and added to the list. The name comparison is case-insensitive.

  - *Date* (string) – the UTC data and time when the model was installed. The format should be YYYY/MM/DD HH:MM:SS. This is not used for anything except being stored with the model.

  - *MinAzLimit* (float) – minimum azimuth for which this model is appropriate. If the telescope ever moves beyond this limit, tracking is stopped and the Status.MountModelLimit flag will be set. Set to zero to specify no limit.

  - *MaxAzLimit* (float) – maximum azimuth for which this model is appropriate. If the telescope ever moves beyond this limit, tracking is stopped and the Status.MountModelLimit flag will be set. Set to zero to specify no limit.

  - *MinElLimit* (float) – minimum elevation for which this model is appropriate. If the telescope ever moves beyond this limit, tracking is stopped and the Status.MountModelLimit flag will be set. Set to zero to specify no limit.

  - *MaxElLimit* (float) – maximum elevation for which this model is appropriate. If the telescope ever moves beyond this limit, tracking is stopped and the Status.MountModelLimit flag will be set. Set to zero to specify no limit.

  - *Terms* (array of MountModelTermStruct) – this field defines the formula and value for each term comprising the mount model. The MountModelStruct contains the following fields:

    - *Name* (string) – short descriptive name for this term

    - *Formula* (string) – the formula for this term. See §4.2.2.2.4 for details

    - *Value* (float) – value of this term, *i.e.*, its maximum contribution in arc seconds.

- *EnableSunAvoidance* and *DisableSunAvoidance* indicates to the telescope server whether to avoid the sun.

  The Sun avoidance function is used to protect the telescope optics from direct exposure to sunlight. If the telescope control system determines that the telescope may be too close to the Sun, it will disable all telescope motion, including tracking, homing, stopping or parking. The telescope can only be moved again if Sun avoidance is disabled or the Sun moves away from the telescope or below the horizon.

  The telescope control system also considers that the telescope may be too close to the Sun if either the telescope position or the Sun position is unknown. The telescope position is unknown until all the telescope axes are successfully homed. The Sun position is unknown if there is a telescope clock error or the telescope site coordinates are invalid.

4.2.2.2.4 Mount Model Formulas

The terms in the mount model are defined using a simple symbolic notation. In this way, the tool allows for a wide range of possible terms, well beyond the typical set employed in standard analysis packages.

Each term uses any of several “input” values to compute any of several “output” values. The input values may be combined using the usual arithmetic operators and expressions, albeit with an abbreviated syntax that is designed for easy parsing. Inputs are the actual values; outputs are proportional to the *changes* in those values. All angles are in radians.

Table 3 Mount model Input Values Notation

Input Value	Notation
`Azimuth`	A
`Sky Azimuth`	S
`Elevation`	E
`Hour Angle`	H
`Declination`	D

Table 4 Mount Model Input Values Notation

Output Value	Notation
`Azimuth`	A
`Zenith Angle`	Z
`Elevation`	E

Output Value	Notation
Hour Angle	H
Declination	D
Parallactic Angle	P
Latitude	L
Pi	p

Table 5 Mount Model Operators Notation

Function	Notation
Constant	1
Separator	;
Assignment	=
Addition or unary plus	+
Subtraction, or unary minus	-
Multiplication	*
Division	/
Exponent	^
Begin expression	(
Close expression	)
Sine Function	s
Cosine Function	c
Tangent Function	t
Degrees-to-radians Function	r
Modulo Function	m

Multiplication is the assumed operator when two terms are adjacent.

Thus, we have the following examples:

A=1 means that the contribution to the modeled error in altitude from this term is to be proportional to a constant.

A=tEcA;E=-sA means that the contribution to the modeled error in azimuth from this term is to be proportional to the product of the tangent of elevation and cosine of altitude; and that the contribution in elevation is to be proportional to the additive inverse of the sine of altitude.

A=s(2A) means that the contribution to the modeled error in azimuth from this term is to be proportional to the sine of twice the altitude.

The mount model does not account for thermal effects, atmospheric refraction, offsets or non-deterministic effects such as seeing or drive hysteresis not does it correct for any moveable stages other than the gimbal. Separate corrections are implemented explicitly for these effects. Each instrument port may have its own mount model, or many ports may share a model.

Startup and Shutdown

Under normal operation, operators should not have to start and stop the TelescopeServer since it would be managed by the system management interface or scripts. If the TelescopeServer is started manually, the following command line arguments are available:

- /SIMULATE - Instructs the server to start in 'Simulate' mode, overriding the setting in the configuration file. This is useful for testing the communications with client software. However, the DIOP is not simulated, so status changes may not occur as they do on the real system.
- /TRACE - Instructs the server to start with 'Trace' mode enabled. This causes the server to log additional debugging information. This switch overrides the setting in the configuration file. The *DebugTrace* command may still be used to turn off trace mode after the server has started. Trace mode should not be enabled under normal operation.

4.2.2.2.5 Framework 2.1.x Parameters

The Framework 2.1.x version of the server may require the following additional parameters. Note that case and order must be preserved:

- -name <ServerName> - The <ServerName> field indicates the name of the server as it is registered with the Framework system. This is usually "DIOServer," however it may be changed if the system has been configured to register the server under a different name.
- -location <Location> - The <Location> field indicates the logical location of the server as it is registered with the Framework system. This is usually "Telescope," however it may be changed if the system has been configured to register the server with a different location.
- -isManaged <ManagedFlag> - The <ManageFlag> is either 'true' or 'false' and indicates if the server is being managed by the system management interface.

4.2.2.2.6 Framework 2.0.x 'Local' Configuration File

The Framework 2.0.x version of the server must have a 'local' configuration file in the same directory as the executable, with the following example contents:

```
LocalConfig
{
    Name      STRING
    Location  STRING
}

@

LocalConfig
{
    Name      "TelescopeServer"
    Location  "Telescope"
}
```

This file should have the same name as the server, but with a ".dmx" extension, i.e. TelescopeServer.dmx. This provides the Framework system with the same registration information as the -name and -location command line parameters do for the 2.1.x version of the Framework.

4.2.2.2.7 System Configuration Notice

The EOS Framework system is made up of a handful of communication, configuration, and operational servers. Several prerequisite servers must be running and properly configured on the observatory network for the TelescopeServer to detect and register itself with the system. Several of these system servers also require configuration to expect the registration attempts by the TelescopeServer. The configuration and operation of these servers is outside the scope of this document.

4.2.2.2.8 Shutdown

The TelescopeServer may be shutdown a number of ways. Typically it is shutdown by either the system management interface or by a shutdown script, which would shutdown the entire system. The TelescopeServer also responds to a 'Break' signal such as control-c if running in a console and will shutdown if such a signal is encountered. Operating system dependant 'kill' signals can be used to shutdown a background instance, assuming adequate permissions and access to the host computer.

4.2.3 GPS SERVER

The GPS Time Server (GPSServer) is a gateway application that adapts the time and position data from the Symmetricom/TrueTime XL-AK GPS Time Receiver into DML structure suitable for publication and retrieval by EOS Framework client applications.

The GPSServer is useful for synchronizing time dependant data with UTC time, provided by the GPS network. To communicate with the GPS Receiver, the GPSServer uses a serial (RS-232) interface between the host computer and the receiver.

The GPSServer also has a simulation mode that does not require access to the GPS Receiver device. The simulation mode allows developers to test the communications between the GPSServer and client applications. The simulation utilizes a configured position and the host computer's built-in clock.

The GPSServer operates by repeatedly polling the GPS receiver for UTC time. The time is published whenever the seconds field changes, effectively publishing once a second. The last sub-second time can be queried using the *GetTime* command. Additional auxiliary data are polled less frequently. This includes GPS position and satellite quality codes.

The configuration and schema files are on the dome computer in c:\eos\bin\servers\ConfigServer\ConfigFiles\Telescope.

4.2.3.1 Configuration and Setup

The GPSServer uses standard DMX configuration files. Some familiarity with the EOS DMX configuration system is assumed. The GPSServer must be restarted to see any changes made to the configuration file. Editing configuration should be performed only by system engineers or administrators; however the configuration file may contain useful information for operators.

4.2.3.1.1 Application

The GPSServer supports the standardized *ApplicationConfig* structure, common to most EOST servers. This allows administrators to set the following common parameters:

- *Simulate* (boolean) - This allows administrators to enable or disable the simulation mode via the configuration file. This can also be enabled when running the server via a command line argument. This is typically set to false.
- *Trace* (boolean) - This allows administrators to enable or disable the trace mode via the configuration file. This can also be enabled using the DML command *DeviceServer.Commands.DebugTrace* { } and via a command line argument. This turns

on additional trace messages recorded in the server's log file. These are useful for debugging purposes. Due to the number of messages generated by this mode, it is not recommended that this be enabled during normal operation.

- *ScanRate* (float: seconds) - This allows administrators to adjust the duty cycle period that the server uses to query the receiver for auxiliary data. This is typically set to 60 seconds, since the auxiliary data does not change frequently.
- *TriggerSleepTime* (float: seconds) - Optionally adds a delay to the thread responsible for polling the GPS receiver for the current time. The value defaults to -1 if not set in the configuration file. A negative value disables the sleep call. This is typically set to 0.1 seconds to reduce the number of operations performed at a time.

4.2.3.1.2 SerialPort

The *SerialPort* section of the configuration file allows an administrator to setup the serial connection to the GPS Receiver. The host computer must be connected to the RS-232 serial port on the GPS Receiver. The hardware section implements a *SerialPortConfig* structure, which has the following parameters

- *PortNumber* (integer) - A number that identifies the serial port on the host computer, which the receiver is connected. Under Windows this number identifies the COM port to use. Under Linux this number identifies the /dev/ttyS* device file to use.
- *BaudRate* (integer: bps) - The transmission rate for the serial connection. The GPS Receiver must be set to match this transmission rate. This is typically set to 19,200 bps.
- *DataBits* (integer: 7 or 8) - The size of a 'byte' of data for the serial connection. The GPS Receiver must be set to match this setting. This is typically set to 7.
- *StopBits* (integer: 1 or 2) - An identifier for the style of bits used for synchronizing the serial port. The GPS Receiver must be set to match this setting. This is typically set to 1.
- *Parity* (string: "None", "Odd", or "Even") - Identifies a serial error detection scheme. The GPS Receiver must be set to match this setting. This is typically set to "Even".
- *FlowControl* (string: "None", "Xon/Xoff", or RTS/CTS") - A setting that identifies hardware or software flow control settings. This is typically set to "None".
- *Timeout* (integer: milliseconds) - A timeout value for serial operations. This is typically allowed to default to 10000 ms.

4.2.3.1.3 SimulationPosition

The SimulationPostion section of the configuration file allows an administrator to specify the position reported by the server while running in 'Simulate' mode. The *SimulationPosition* section implements a *PositionConfig* structure with the following parameters:

- *Latitude* (float: degrees) - The Geodetic latitude of the simulated position. Typically this is set to the approximate location of the operator.
- *Longitude* (float: degrees) - The Geodetic longitude of the simulated position. Typically this is set to the approximate location of the operator.
- *Height* (float: meters) - The simulated altitude of the simulated position in meters above sea level. Typically this is set to the approximate location of the operator.

4.2.3.2 Schema and Operation

The operation of the GPSServer is reflected by the Schema-GPSServer.dms file. The schema file defines attributes and commands used to operate the GPS device. Attributes are published structures containing status information and can be retrieved by clients either by subscription or by using a status retrieving command.

Commands and attributes are sent and received using the EOS Control System Client API. The Device Browser application allows access to this interface without needing to write a client application, however operators should rarely need to manipulate servers using the schema directly. Instead, they should use high-level user interfaces, such as the TelescopeInterface, which may present the attributes and commands in manner that does not resemble the interface described here. Operators should refer to the documentation of the client application.

The GPSServer schema divides operations into top-level and logical devices. The top-level commands and attributes are used to monitor the state of the server itself. The logical devices group similar functions and provide an abstraction of the major operations of the server.

The GPSServer schema file should not be altered. Changes to the schema can only be implemented by updating and releasing a new version of the server.

4.2.3.2.1 ApplicationInfo

The *ApplicationInfoStruct* is a standardized attribute structure that is used by most EOST servers. It contains information about the GPSServer as it is running. The *ApplicationInfoStruct* has the following fields:

- *Version* (string) - A string containing the version number of the GPSServer. The current GPSServer will report "3.3.3.15" if using the 2.1.2 version of the EOS Framework.
- *BuildInfo* (string) - A string containing the date and time that the GPSServer was built. The date string will resemble: "Jul 23 2009 14:57:34."
- *SchemaVersionMajor* (integer) - A number representing the major schema version of the server. This will correspond with the X in the *__Schema_Version_Major__X* field of the *__Schema_Version__* structure if the server was built using the same schema as it is currently running with.
- *SchemaVersionMinor* (integer) - A number representing the minor schema version of the server. This will correspond with the Y in the *__Schema_Version_Minor__Y* field of the

__Schema_Version__ structure if the server was built using the same schema as it is currently running with.

- *Simulated* (boolean) - A flag that indicates if the server is running in 'Simulate' mode. The flag will be false if the server is operating normally.
- *Trace* (boolean) - A flag that indicates if the server is running with additional tracing turned on. If true, additional debugging information is being logged in the server's log file.
- *Fault* (*EventInfo*) - A structure that indicates why the server may currently be in a fault mode. The server will transition to 'Error' mode when a fault requiring operator intervention has occurred. The fault field will report information about the error that has occurred, however additional investigation, including examining log files may be required to determine the proper course of action. Sending the server the software Reset command will clear the fault field. The *EventInfo* structure contains the following fields:
    - *Reason* (string) - A string representing the likely cause of the error state.
    - *Date* (string) - A string representing the date and time the fault occurred.

4.2.3.2.2 Commands

The GPSServer will respond to several common top-level commands that are common to most EOST servers. All command structures of servers written by EOST will contain two sub-structures: Parameters and Returns. The Parameters structure indicates the command parameters that may be required by the server. The Returns structure indicates the structure of the data that will be returned by the server upon successful operation. The following top level commands are for manipulating and retrieving information about the server:

- *GetApplicationInfo* - Returns an *ApplicationInfo* structure, which matches the structure published in the Attributes section. This command provides an alternate method of retrieving the structure that does not require a subscription.
- *Reset* - If the server is in an Error state, this command will attempt to reset the server and clear the error state. The reset command will cause the server to reinitialize the connection to the GPS Receiver.
- *DebugTrace* - This debug command allows an operator to enable or disable the trace mode of the server while it is running. This turns on additional logging that may be useful for debugging purposes. Debug commands may be removed on production installations.
- *DebugSetFault* - This debug command artificially instigates a server fault. This may be useful for debugging purposes. Debug commands may be removed on production installations.

4.2.3.2.3 ReceiverInfo Device

The ReceiverInfo device provides information about the GPS receiver. The following four fields are provided by the ReceiverInfo device:

- *SoftwareVersion* (string) - The version of the software running on the GPS Receiver. This field can either be subscribed to by clients, or retrieved using the *GetSoftwareVersion* command.
- TimeInfo (*TimeInfoSet* structure) - A structure that contains information about the time settings on the receiver. This field can either be subscribed to by clients, or retrieved using the *GetTimeInfo* command. The following fields are provided by the *TimeInfoSet* structure:
    - *TimeQualityThresholds* (string) - A string that indicates the four time quality thresholds used by the receiver for indicating time quality. The GPSServer sets this value, so it should always be "1000 10000 100000 1000000."
    - *TimeZone* (string) - A string indicating the time zone used by the receiver. This GPSServer sets this value, so it should always be "UTC."
- *AlarmStatus* (string) - A string indicating the receiver alarm status. See the Symetricomm XL-AK documentation for the table of codes used to decipher the alarm status string. This field can either be subscribed to by clients, or retrieved using the *GetAlarmStatus* command.
- *FaultStatus* (string) - A string indicating the receiver fault status. See the Symetricomm XL-AK documentation for the table of codes used to decipher the fault status string. This field can either be subscribed to by clients, or retrieved using the *GetFaultStatus* command.

4.2.3.2.4 TimeData Device

The *TimeData* device provides the current UTC time and gives a quality indication. The *TimeData* device provides the following fields:

- *DateTime* (*DateTimeStruct* structure) - This field provides the UTC time as separate Gregorian unit fields:
    - *Year* (integer) - The four-digit year number.
    - *Month* (integer) - The month of year, 1 though 12.
    - *DayOfYear* (integer) - The day or year, 1 through 366 (depending on leap year).
    - *Day* (integer) - The day of month, 1 through 31 (depending on month).
    - *Hour* (integer) - The hour of day, 0 through 23.
    - *Minute* (integer) - The minutes of the hour, 0 through 59.
    - *Second* (integer) - The second of the minute, 0 through 59 (Leap seconds may cause the second field to exceed 59).
    - *Microsecond* (integer) - The microsecond field of the second. The GPS receiver only provides milliseconds, so this is really milliseconds * 1000. Microseconds is used to be consistent with other servers.

- *TimeQuality* (float: seconds) - An indication of time quality provided by the GPS receiver. The worst-case error is greater than this value, but less than the next threshold. If simulated, the server will report 0.

The *DateTime* field is published once per second. The time can be retrieved sooner using the *GetDateTime* command. The *GetDateTime* command typically returns the most recent polled time from the GPS Receiver. The command also has an optional parameter, *WaitForNext*, which, when true, causes the command to block until the next time poll is performed.

- *JulianDate* (float) - The UTC time above, converted to a double-precision Julian Date. The JulianDate field is similarly only published once per second.
- *WorstCaseError* (float: seconds) - The worst-case time error reported by the GPS receiver due to oscillator drift while not tracking satellites. The error reported while tracking satellites is always 00.000000200 s (0.2 µs). While simulating, the server will report 0. This field is updated by a poll, running at the *ScanRate* period, which is typically one minute. The *GetWorstCaseError* command will return the last error retrieved by the *ScanRate* poll.
- *GetYear* (integer) - Alternate method to get the four digit Gregorian year.

4.2.3.2.5 LocationData Device

The *LocationData* device provides the *AveragePosition* of the GPS Receiver via published attribute and the *GetAveragePosition* command. The average position is updated at the *ScanRate* poll interval. The *AveragePosition* is returned via a *Position* structure with the following fields:

- *Latitude* (float: radians) - The Geodetic latitude reported by the GPS receiver.
- *Longitude* (float: radians) - The Geodetic longitude reported by the GPS receiver.
- *Height* (float: meters) - The altitude reported by the GPS receiver in meters above sea level.
- *Count* (integer) - The total number of position fixes in the average. This number is out of 90,000.

4.2.3.3 Startup and Shutdown

Under normal operation, operators should not have to start and stop the GPSServer since it would be managed by the system management interface or scripts. If the GPSServer is started manually, the following command line arguments are available:

- /SIMULATE - Instructs the server to start in 'Simulate' mode, overriding the setting in the configuration file. This is useful for testing the communications with client software.
- /TRACE - Instructs the server to start with 'Trace' mode enabled. This causes the server to log additional debugging information. This switch overrides the setting in the

configuration file. The *DebugTrace* command may still be used to turn off trace mode after the server has started. Trace mode should not be enabled under normal operation.

4.2.3.3.1 Framework 2.1.x Parameters

The Framework 2.1.x version of the server may require the following additional parameters. Note that case and order must be preserved:

- -name <ServerName> - The <ServerName> field indicates the name of the server as it is registered with the Framework system. This is usually "GPSServer," however it may be changed if the system has been configured to register the server under a different name.
- -location <Location> - The <Location> field indicates the logical location of the server as it is registered with the Framework system. This is usually "Telescope," however it may be changed if the system has been configured to register the server with a different location.
- -isManaged <ManagedFlag> - The <ManageFlag> is either 'true' or 'false' and indicates if the server is being managed by the system management interface.

4.2.3.3.2 Framework 2.0.x 'Local' Configuration File

The Framework 2.0.x version of the server must have a 'local' configuration file in the same directory as the executable, with the following example contents:

```
LocalConfig
{
    Name      STRING
    Location  STRING
}

@

LocalConfig
{
    Name      "GPSServer"
    Location  "Telescope"
}
```

This file should have the same name as the server, but with a ".dmx" extension, i.e. GPSServer.dmx. This provides the Framework system with the same registration information as the -name and -location command line parameters do for the 2.1.x version of the Framework.

4.2.3.3.3 System Configuration Notice

The EOS Framework system is made up of a handful of communication, configuration, and operational servers. Several prerequisite servers must be running and properly configured on the observatory network for the GPSServer to detect and register itself with the system. Several of these system servers also require configuration to expect the registration attempts by the GPSServer. The configuration and operation of these servers is outside the scope of this document.

4.2.3.3.4 Shutdown

The GPSServer may be shutdown a number of ways. Typically it is shutdown by either the system management interface or by a shutdown script, which would shutdown the entire system. The GPSServer also responds to a 'Break' signal such as control-c if running in a console and will shutdown if such a signal is encountered. Operating system dependant 'kill' signals can be used to shutdown a background instance, assuming adequate permissions and access to the host computer.

4.2.3.4 Potential Year Rollover Issues

The GPS time signal does not contain a year indicator. The GPS receiver utilizes several strategies to determine the correct year. Upon initialization the GPSServer queries the receiver and sets its internal year variable. Subsequent roll-overs of the 'Day of Year' field in the time signal will cause the GPSServer to increment its internal year variable. Periodically the GPSServer will retrieve the year field from the receiver and compare its internal variable to the receiver's. If the year does not match, the GPSServer will use the later year and log an error message in its log file.

4.2.4 TEMPERATURE SENSOR SERVER

The i8000TemperatureServer is a gateway application that adapts the temperature data from a network of temperature controllers and RTD temperature sensors into DML structure suitable for publication and retrieval by EOS Framework client applications.

The i8000TemperatureServer also has a simulation mode that does not require access to the temperature controller network. The simulation mode allows developers to test the communications between the i8000TemperatureServer and client applications. The simulation utilizes a configured temperature set-point and a pseudo-random offset for each sensor.

The configuration and schema files are on the dome computer in c:\eos\bin\servers\ConfigServer\ConfigFiles\Telescope.

4.2.4.1 Temperature Controllers and Sensors

The temperature controller network is comprised of one or many temperature controllers that exist on the telescope network. Temperature controllers are either the ICPDAS I-8431 or I-8831 Ethernet embedded controller. The I-8431 supports four I/O modules, while the I-8831 supports eight. Both controllers share the same head module and differ only by the backplane. ICPDAS also offers a choice of 80 MHz or 40 MHz clock speed variants, but for temperature sensing, either is sufficient. The controller must be programmed with the EOST authored I-8000 Controller Basic Firmware (XBASIC.EXE) and connected to the telescope network via the onboard Ethernet port. The controller also requires a 24V dc supply rated at or above 1 amp. DC power is typically supplied by the 24V supply in the Digital IO Planar (DIOP) of the telescope rack. EOST refers to the controllers generically as I-8000 controllers. The position in the controller where a module may be installed is referred to as a slot.

To read the RTD sensors, the temperature controller must be populated with I-87K input modules. EOST typically utilizes two modules: I-87013 or I-87015P depending on installation envelope size and number of sensors. The I-87013 module supports 4 RTD input channels and has a 'low' profile. The I-87015P module supports 7 RTD input channels. Both modules, when properly configured and wired will compensate for cable length. EOST has telescope installations which utilize i-87015 modules that do not compensate for cable length. If these modules are installed, the installation cable length must be approximated when measuring the sensors for calibration.

Utilizing the 7 channel RTD modules and an 8 slot controller yields a maximum of 56 sensors per controller. More sensors can be utilized by adding additional controllers to the network.

Temperature sensors are Platinum 100 RTDs, with α = 0.00385 and range from -100°C to 100°C. EOST Typically uses Omega RTD-830 surface mount sensors and Omega RTD-805 air temperature sensors.

The I-87013 modules should be configured to match the RTD type and alpha code. The programming code is: `%0000200A00`. This is the default configuration. The I-87015 modules should be configured per channel. The programming code is: `$007CiR20`, where 'i' is the channel number 0-6. This is the default setting. Note that older I-87015 modules do not support this setting. This command may be sent to the controller using the server's *DebugSendRawCommand* command.

EOST typically provides one temperature controller and 6-8 sensors on each telescope at the following locations:

- One surface sensor on each truss leg or every other truss leg (2-4 sensors)
- One surface and one air sensor on or near the secondary mirror (2 sensors)
- One surface and one air sensor on or near the primary mirror (2 sensors)

EOST may provide additional sensors as required by contractual agreements. Customers may add additional sensors and controllers to the network as necessary, however the customer is responsible for any hardware in excess of the contractual obligations.

Sensors are typically calibrated at the EOST factory prior to installation on the telescope. To calibrate sensors, any existing calibration must be removed from the server's configuration file and the server restarted. All sensors are then placed in a water and crushed ice solution and allowed to athermalize for several minutes. The temperature of the ice bath is recorded by each sensor and with a calibrated digital thermometer. The sensors are then removed from the ice bath and placed in a warm water bath. Again the temperature of the warm water bath is recoded by each sensor and with a calibrated digital thermometer. With two data points for each sensor a linear calibration can be calculated as a gain (scalar) and offset pair. The i8000TemperatureServer provides a mechanism for entering the gain and offset pair for each sensor into the configuration file. The following algorithm may be used to calculate the gain and offset for calibration, where, for each sensor T is the temperature measured with the calibrated digital thermometer, t is the temperature measured with the RTD sensor, G is the gain, and O is the offset:

$$G = (T_1 - T_0)/(t_1 - t_0)$$
$$O = T_0 - (G \cdot t_0)$$

If a gain and offset pair are configured for a sensor, the server will calculate the 'true' temperature based on the following formula, where, for each sensor T is the 'true' temperature, t is the temperature measured with the RTD sensor, G is the gain, and O is the offset:

$$T = t \cdot G + O$$

4.2.4.2 Configuration and Setup

The i8000TemperatureServer uses standard DMX configuration files. Some familiarity with the EOS DMX configuration system is assumed. See Appendix A for an example i8000TemperatureServer.dmx file. The i8000TemperatureServer must be restarted to see any changes made to the configuration file. Editing configuration should be performed only by system engineers or administrators; however the configuration file may contain useful information for operators.

This document refers the 'Network' configuration file. In order for the i8000TemperatureServer to function properly the configuration file must reside in a directory determined by the Framework's configuration system. Additional configuration files may need to be edited in order to set up the i8000TemperatureServer before its first use.

4.2.4.2.1 Application

The i8000TemperatureServer supports the standardized *ApplicationConfig* structure, common to most EOST servers. This allows administrators to set the following common parameters:

- *Simulate* (boolean) - This allows administrators to enable or disable the simulation mode via the configuration file. This can also be enabled when running the server via a command line argument. This is typically set to false.
- *Trace* (boolean) - This allows administrators to enable or disable the trace mode via the configuration file. This can also be enabled using the DML command *DeviceServer.Commands.DebugTrace* { } and via a command line argument. This turns on additional trace messages recorded in the server's log file. These are useful for debugging purposes. Due to the number of messages generated by this mode, it is not recommended that this be enabled during normal operation.

4.2.4.2.2 Times

The *Times* section of the configuration file implements a *SensorTimeInfo* structure with the following parameters:

- *ScanRate* (float: seconds) - This allows administrators to adjust the duty cycle period that the server uses to query the temperature controller for new sensor readings and publishes the new readings. This is typically set to 5 seconds.
- *Timeout* (float: seconds) - Sets the wait time for socket operations. If a read operation exceeding the *Timeout* duration the server will assume that the controller is unreachable and set an error condition. This is typically 15 seconds.

4.2.4.2.3 TempsLog

The *TempsLog* section of the configuration file allows administrators to control the logging of temperature data. If enabled, temperature data is logged to a file, separately from the server's operational log in a Comma Separated Value (CSV) format. Simulated data will not be logged to the file. Each time the server is started, a header is printed to the log file with the name of the sensors being logged. The *TempsLog* section implements a *TempsLogInfo* structure with the following parameters:

- *Enabled* (boolean) - Determines if the temperature logging feature is enabled. This is typically true.
- *Filename* (string) - Determines the naming convention for files created by the temperature logger. Under Windows the value is usually: "C:\EOST\Data\TemperatureData\%Y%M%D-i8000TemperatureServer.csv." Under Linux the value us usually: "/EOST/Data/TemperatureData/%Y%M%D-i8000TemperatureServer.csv." The "%Y%M%D" portion of the string is replaced with the year, month, and day respectively. An example filename is: "20090728-i8000TemperatureServer.csv." If the filename parameter is not specified, the logger will write the files to the directory where the server was executed from.

4.2.4.2.4 ControllerList

The *ControllerList* configuration allows administrators to specify the temperature controllers that make up the controller network. Each controller structure in the list allows the operator to specify the individual sensors, whose temperature will be collected by the server. Each controller is an instance of an *I8000Config* structure, with the following parameters:

- *Controller* (integer) - A number that is used to identify the controller. This can be any integer, but must only appear once in the controller list. Typically, the first controller is 0.
- *Enabled* (boolean) - This flag instructs the server to ignore the controller without requiring that the controller's configuration be removed. This may be used if a specific controller is malfunctioning or is temporarily disabled or unreachable. Typically all controllers are enabled.
- *IPAddresss* (string) - The IPv4 address of the temperature controller.
- *Port* (integer) - The network port of the temperature controller. The network port is typically 10,000 for all I-8000 Ethernet devices.
- *SensorList* (array of *SensorConfig*) - The list of individual sensors associated with this controller. Each sensor is configured using a *SensorConfig* structure with the following parameters:
  - *Label* (string) - A textual description, such as: "Center Section Air." This string is used to identify the sensor on the telescope.
  - *Groups* (array of string) - A list of sensor groups that the sensor belongs to. An example group is, "Truss." These are purely logical designations used to associate sensors with one another relative to telescope or observatory components. The "Truss" group has a specific connotation, since this group is used to identify the sensors to be averaged to calculate the *AverageTrussTemperature* attribute.
  - *Slot* (integer) - The position of the RTD module in the I-8000 controller. Positions are numbered from 0 through 7 on an 8 slot controller, or 0 through 3 on a 4 slot controller. This identifies the module where the sensor is installed.
  - *Channel* (integer) - The position of the RTD in the sensor module. Channels are numbered 0 through 3 for 4 channel modules like the I-87013 and 0 through 6 for six channel modules like the I-87015.
  - *Enabled* (boolean) - Instructs the server to ignore this specific sensor. This may be useful if the sensor is malfunctioning or disconnected.
  - *Simulated* (boolean) - Instructs the server to use the simulated temperature for this specific sensor. This parameter overrides the server simulated setting so the sensor value will be simulated even if the server is not in simulated mode.
  - *SimulatedTemperature* (float) - The temperature that will be the basis for this sensor in degrees C if the sensor or server is simulated. The simulated temperature reported will also have a small pseudo-random offset.
  - SimulatedStatus (string) - The status that will be reported by the server for this sensor if it is simulated.

- *Calibration* (*SensorCalibrationConfig*) - The optional calibration parameters for the sensor. The default values causes the server to report the sensor as it was read. The SensorCalibrationConfig structure has the follwing parameters:
  - *Gain* (float) - The scale of the sensor error. The default value is 1.0.
  - *Offset* (float) - The offset of the sensor error. The default value is 0.0.

4.2.4.3 Schema and Operation

The operation of the i8000TemperatureServer is reflected by the Schema-i8000TemperatureServer.dms file. The schema file defines attributes and commands used to operate the temperature controller network. Attributes are published structures containing status information and can be retrieved by clients either by subscription or by using a status retrieving command.

Commands and attributes are sent and received using the EOS Control System Client API. The Device Browser application allows access to this interface without needing to write a client application, however operators should rarely need to manipulate servers using the schema directly. Instead, they should use high-level user interfaces, such as the TelescopeInterface, which may present the attributes and commands in manner that does not resemble the interface described here. Operators should refer to the documentation of the client application.

The i8000TemperatureServer schema divides operations into top-level and a logical device. The top-level commands and attributes are used to monitor the state of the server itself. The logical device provides an abstraction of the major operations of the temperature network.

The schema file should not be altered. Changes to the schema can only be implemented by updating and releasing a new version of the server.

4.2.4.3.1 ApplicationInfo

The *ApplicationInfoStruct* is a standardized attribute structure that is used by most EOST servers. It contains information about the i8000TempeartureServer as it is running. The *ApplicationInfoStruct* has the following fields:

- *Version* (string) - A string containing the version number of the i8000TemperatureServer. The current GPSServer will report "3.3.3.18" if using the 2.1.2 version of the EOS Framework.
- *BuildInfo* (string) - A string containing the date and time that the i8000TemperatureServer was built. The date string will resemble: "Jul 23 2009 14:57:34."
- *SchemaVersionMajor* (integer) - A number representing the major schema version of the server. This will correspond with the X in the *__Schema_Version_Major__X* field of the *__Schema_Version__* structure if the server was built using the same schema as it is currently running with.

- *SchemaVersionMinor* (integer) - A number representing the minor schema version of the server. This will correspond with the Y in the *__Schema_Version_Minor__Y* field of the *__Schema_Version__* structure if the server was built using the same schema as it is currently running with.
- *Simulated* (boolean) - A flag that indicates if the server is running in 'Simulate' mode. The flag will be false if the server is operating normally.
- *Trace* (boolean) - A flag that indicates if the server is running with additional tracing turned on. If true, additional debugging information is being logged in the server's log file.
- *Fault* (*EventInfo*) - A structure that indicates why the server may currently be in a fault mode. The server will transition to 'Error' mode when a fault requiring operator intervention has occurred. The fault field will report information about the error that has occurred, however additional investigation, including examining log files may be required to determine the proper course of action. Sending the server the software Reset command will clear the fault field. The *EventInfo* structure contains the following fields:
    - *Reason* (string) - A string representing the likely cause of the error state.
    - *Date* (string) - A string representing the date and time the fault occured.

4.2.4.3.2 Server Commands

The i8000TemperatureServer will respond to several common top-level commands that are common to most EOST servers. All command structures of servers written by EOST will contain two sub-structures: Parameters and Returns. The Parameters structure indicates the command parameters that may be required by the server. The Returns structure indicates the structure of the data that will be returned by the server upon successful operation. The following top level commands are for manipulating and retrieving information about the server:

- *GetApplicationInfo* - Returns an *ApplicationInfo* structure, which matches the structure published in the Attributes section. This command provides an alternate method of retrieving the structure that does not require a subscription.
- *Reset* - If the server is in an Error state, this command will attempt to reset the server and clear the error state. The reset command will cause the server to reinitialize the connection to the temperature controllers.
- *DebugTrace* - This debug command allows an operator to enable or disable the trace mode of the server while it is running. This turns on additional logging that may be useful for debugging purposes. Debug commands may be removed on production installations.
- *DebugSetFault* - This debug command artificially instigates a server fault. This may be useful for debugging purposes. Debug commands may be removed on production installations.

4.2.4.3.3 *i8000Temperature Attributes*

The *i8000Temperature* device publishes the following attributes:

- *Connected* (boolean) - Flag that indicates that all enabled temperature controllers are connected to the server.
- *SensorData* (array of *SensorValueStruct*) - An array containing information and the temperature of each enabled sensor. The sensor data is updated every *ScanRate* seconds. The *SensorValueStruct* has the following parameters:
  - *Label* (string) - The textual description of the sensor. This is assigned in the server's configuration file. An example label is, "Center Section Air."
  - *Groups* (array of string) - An array indicating the groups the sensor is a member of. This is assigned in the server's configuration file. An example group is, "Truss."
  - *Temperature* (float: degrees Celsius) - The temperature value of the sensor after calibration correction is applied to the last temperature reading from the controller.
  - *Status* (string) - Sensor status indicator. This either reports "Normal" or "Simulated" if the simulated temperature is being reported.
  - *Simulated* (boolean) - Additional flag indicating that the temperature value is simulated.
- *TimeStamp* (*DateTimeStruct*) - A UTC time stamp recorded with each collection interval, after all of the enabled sensors have been queried. The *DateTimeStruct* has the following parameters:
  - *Year* (integer) - The four-digit year number.
  - *Month* (integer) - The month of year, 1 though 12.
  - *DayOfYear* (integer) - The day or year, 1 through 366 (depending on leap year).
  - *Day* (integer) - The day of month, 1 through 31 (depending on month).
  - *Hour* (integer) - The hour of day, 0 through 23.
  - *Minute* (integer) - The minutes of the hour, 0 through 59.
  - *Second* (integer) - The second of the minute, 0 through 59 (Leap seconds may cause the second field to exceed 59).
  - *Microsecond* (integer) - The microsecond field of the second.
- *AverageTrussTemperature* (float: degrees Celcius) - The average of the temperature readings of any enabled sensors that belong to the group "Truss."

4.2.4.3.4 *i8000Temperature Commands*

The *i8000Temperature* device provides the following commands:

- *GetSensors* - Returns an array containing the most recent temperature values for each sensor and the timestamp indicating when they were collected. The command optionally takes a *Group* string as a parameter. If the *Group* parameter is specified the command will only return the values for sensors that belong to the group specified, otherwise the temperature value for all sensors is returned. The command will return an empty list if the group name does not exist. The command returns the following structures
    - *TimeStamp* (*DateTimeStruct*) - A UTC time stamp recorded with each collection interval, after all of the enabled sensors have been queried. The *DateTimeStruct* has the following parameters:
        - *Year* (integer) - The four-digit year number.
        - *Month* (integer) - The month of year, 1 though 12.
        - *DayOfYear* (integer) - The day or year, 1 through 366 (depending on leap year).
        - *Day* (integer) - The day of month, 1 through 31 (depending on month).
        - *Hour* (integer) - The hour of day, 0 through 23.
        - *Minute* (integer) - The minutes of the hour, 0 through 59.
        - *Second* (integer) - The second of the minute, 0 through 59 (Leap seconds may cause the second field to exceed 59).
        - *Microsecond* (integer) - The microsecond field of the second.
    - *Sensors* (array of *SensorDataStruct*) - An array containing the temperature value for each sensor. The SensorDataStruct has the following fields:
        - *Label* (string) - The textual description of the sensor. This is assigned in the server's configuration file. An example label is, "Center Section Air."
        - *Status* (string) - Sensor status indicator. This either reports "Normal" or "Simulated" if the simulated temperature is being reported.
        - *Temperature* (float: degrees Celsius) - The temperature value of the sensor after calibration correction is applied to the last temperature reading from the controller.
- *GetInfo* - Returns an array containing static information about each sensor. This information comes from the sensor's configuration in the server's configuration file. The command takes no parameters. The *Sensors* array the command returns contains a *SensorInfoStruct* for each sensor with the following parameters:
    - *Label* (string) - The textual description of the sensor. This is assigned in the server's configuration file. An example label is, "Center Section Air."
    - *Groups* (array of string) - An array indicating the groups the sensor is a member of. This is assigned in the server's configuration file. An example group is, "Truss."
- *DebugSendRawCommand* - Sends a command code to the temperature controller. Note that debug commands may be removed on production installations. Operators

should not send command codes to the controller. Command codes can be used top operate features of the temperature controller not exposed by the server. Module configuration codes can be sent using this command. Module command codes are documented in the ICPDAS documentation. The command has the following parameters:

- *Controller* (integer) - The configured unique number that identifies the controller on the network. This is configured in the server's configuration file.
- *Slot* (integer) - This optionally prepends the slot number onto the control string before sending it to the controller.
- *Command* (string) - The command code to send to the controller. An example command code, "%0000200A00" sets up an I-87013 module.

The command returns the following:

- *Result* (string) - The resulting response from the temperature controller. The example command, "%0000200A00," when sent to an I-87013 module, will return, "!02," which indicates success.

4.2.4.4 Startup and Shutdown

Under normal operation, operators should not have to start and stop the GPSServer since it would be managed by the system management interface or scripts. If the GPSServer is started manually, the following command line arguments are available:

- /SIMULATE - Instructs the server to start in 'Simulate' mode, overriding the setting in the configuration file. This is useful for testing the communications with client software.
- /TRACE - Instructs the server to start with 'Trace' mode enabled. This causes the server to log additional debugging information. This switch overrides the setting in the configuration file. The *DebugTrace* command may still be used to turn off trace mode after the server has started. Trace mode should not be enabled under normal operation.

4.2.4.4.1 Framework 2.1.x Parameters

The Framework 2.1.x version of the server may require the following additional parameters. Note that case and order must be preserved:

- -name <ServerName> - The <ServerName> field indicates the name of the server as it is registered with the Framework system. This is usually "i8000TemperatureServer," however it may be changed if the system has been configured to register the server under a different name.
- -location <Location> - The <Location> field indicates the logical location of the server as it is registered with the Framework system. This is usually "Telescope," however it may be changed if the system has been configured to register the server with a different location.

- -isManaged <ManagedFlag> - The <ManageFlag> is either 'true' or 'false' and indicates if the server is being managed by the system management interface.

4.2.4.4.2 Framework 2.0.x 'Local' Configuration File

The Framework 2.0.x version of the server must have a 'local' configuration file in the same directory as the executable, with the following example contents:

```
LocalConfig
{
    Name      STRING
    Location  STRING
}

@

LocalConfig
{
    Name      "i8000TemperatureServer"
    Location  "Telescope"
}
```

This file should have the same name as the server, but with a ".dmx" extension, i.e. GPSServer.dmx. This provides the Framework system with the same registration information as the -name and -location command line parameters do for the 2.1.x version of the Framework.

4.2.4.4.3 System Configuration Notice

The EOS Framework system is made up of a handful of communication, configuration, and operational servers. Several prerequisite servers must be running and properly configured on the observatory network for the i8000TemperatureServer to detect and register itself with the system. Several of these system servers also require configuration to expect the registration attempts by the i8000Temperature. The configuration and operation of these servers is outside the scope of this document. Refer to the system maintenance manual for information about setup and operation of the Framework system.

4.2.4.4.4 Shutdown

The i8000TemperatureServer may be shutdown a number of ways. Typically it is shutdown by either the system management interface or by a shutdown script, which would shutdown the entire system. The i8000TemperatureServer also responds to a 'Break' signal such as control-c if running in a console and will shutdown if such a signal is encountered. Operating system dependant 'kill' signals can be used to shutdown a background instance, assuming adequate permissions and access to the host computer.

4.3 LOW-LEVEL TOOLS

These tools make a direct connection to the PMAC servo controller without using the Framework. Therefore, great care must be taken because the safety checks provided by the core telescope processes are *not* in force. These tools may be operated at the same time as themselves and other framework processes are running but they are not aware of each other and there is no arbitration, thus which ever tool makes the last command will take effect.

4.3.1 PEWIN

This tool is provided by Delta-Tau as a development and troubleshooting aid for their line of PMAC motion controllers. EOST uses it extensively during telescope construction as each axis is built and brought online. As the axes become mature, more of the control is delegated to the framework processes and the use of PEWIN becomes less common until finally when a system is delivered it is no longer necessary at all under normal circumstances. However, it can be useful when unusual things happen or to gain insight into the servo control of an axis if desired.

For more information, please refer to the following documents available from Delta-Tau:

- Software Reference Manual, Turbo PMAC, 3Ax-01.937-xSxx, November 25, 2008
- User Manual, Turbo PMAC, 3Ax-602264-TU, September 12, 2008
- PMAC Quick Reference, 3A0-PMACQR-xPRx, December 3, 2004

Note well that PEWIN allows direct communication with the PMAC motion controller. Very intimate knowledge of the control system is required to accomplish useful work with this tool. Errors can lead to erratic motion, broken hardware and even injury. Be careful when using PEWIN.

PEWIN can be opened by double-clicking its icon found on the desktop. It is document based and so has its own area for managing subordinate windows.

The window most often used is the command window. This is normally already open by default in the upper left of the PEWIN application. This window is used to type commands directly to PMAC. The window is arranged as a one-line text entry field at the bottom and a scrollable read-only history list of previous commands in the remainder of the window. Click the cursor anywhere in the command window then type desired PMAC commands followed by Enter.

PMAC has dozens of commands and hundreds of registers. It is beyond the scope of this document to describe them, even in cursory fashion. Nevertheless, we include in Appendix A a very brief list of a few common commands. The axis assignments listed in Table 1 are the numbers used for the *xx* values in several register addresses and for the *#n* axis address selector.

For example, to rotate the telescope in azimuth at modest speed, one might enter the following commands:

```
I0922 = 500
#9
J+
```

The source code for the PMAC firmware code is on the TCC computer in C:\EOS\Firmware\PMAC\ALLFILES.PMC. This file can be loaded into the PMAC using the following steps:

1. In the PEWIN command line, type $$$*** to reset PMAC to factory defaults
2. Use File->Download to browse to the desired PMC file and download.
3. If the download says 0 Errors, it compiled and downloaded successfully.
4. Type SAVE to store the new code permanently in EEPROM, if desired
5. Type $$$ to reset the PMAC and start running the new code.

Note the above steps load the new code into nonvolatile storage such that it becomes the default whenever power is cycled on the PMAC. To load it temporarily into RAM only, such as for a temporary change, then skip the SAVE command in step 4.

4.3.2 AXCON

AxCon was written by EOST as an aid to evaluating and improving the tuning of each servo axis. The tool provides a convenient means for making small direct moves on each axis, inspecting the state of each axis, and inspecting and modifying the tuning parameters for each axis. *It can not be overemphasized that great care must be taken when modifying the tuning values for an axis. Incorrect values can lead to erroneous operation, instabilities and physical damage to the telescope structure.* It is harmless and acceptable to just run the program passively at any time in order to monitor the performance of an axis.

AxCon is written in Qt and uses the Servo class from the telescope control framework for all PMAC I/O. Therefore, the EOST PLCs must be running in the PMAC in order to use AxCon and the usual Qt dll runtime files (QtCore4.dll and QtGUI4.dll) must be available.

4.3.2.1 Configuration file

AxCon requires a configuration file unique to each telescope. The file basically contains information as to which motors and encoders are used for each axis, and the scaling of the encoders in terms of convenient units such as degrees or microns. After the program is started, the first step must be to use the File menu to read a configuration file; or the file may be specified as a command line argument when the program is invoked. The latter is performed by the handy shortcut on the desktop for running AxCon.

A representative configuration file for the Lick APF is shown in Appendix B. Comments begin at # and continue to the end of a line. Each axis is described using a fixed set of variables, each of which is prefixed with a unique integer and an underscore. The prefix integers have no meaning other than to group similar variables for an axis but must be in the range 0..31. The variables are described in Table 6

Table 6 AxCon Config File Variables

Variable	**Function**
Motname	This is a short string that is used as a tab heading in the GUI. It has no prescribed meaning, but it seems handy to include some indication of which PMAC axis corresponds to this name.
Enc1	This variable is in two parts, separated by a comma. The first part is an integer giving the PMAC encoder axis, the second part is a short descriptive name.
Enc2	This is optional and has the same format as Enc1 but refers to a second encoder, if present. If there are additional encoders, add them in a similar fashion using variables named Enc3 and Enc4.
AxisType	This is set to either "rotational" or "linear" depending on the mechanical function of the axis. It also determines the set of choices of units in which the Following Error is displayed. For rotational axes, the display units can be chosen as degrees, arc seconds or raw counts. For linear axes, the units may be millimeters, micrometers or raw counts.
PosScale	This is the number of position encoder counts per degree if the AxisType is rotational or the number of counts per millimeter if AxisType is linear.
InitialPos	This is the default value shown in the GUI for a target jog position. If the AxisType is rotational the units are degrees, else if the AxisType is linear the units are in millimeters.
MaxVel	This is the largest value for velocity AxCon will permit the user to enter. The units are degrees/sec if the AxisType is rotational else millimeters/s if AxisType is linear.

Variable	Function
MaxAcc	This is the largest value for acceleration AxCon will permit the use to enter. The units are degrees/sec/sec if the AxisType is rotational else millimeters/s/s if AxisType is linear.

After a configuration file is successfully read, the program establishes contact with the PMAC PLCs. If this is successful, the program presents each available axis as one of many tabs. Selecting a tab shows xx information and provides control over that axis.

4.3.2.2 Operation

Once AxCon is running with a valid configuration file, the user will see one tab for each axis. In the upper left portion of the tab the user may enter a target position and Jog the axis to that position, using the values specified for maximum velocity and acceleration. For safety reasons, the latter values are automatically clamped to the values in the configuration file. For convenience, a step value may be entered and added to or subtracted from the current position for easy incremental moves or moves back and forth. The units in this section are always degrees for rotational axes, or millimeters for linear axes.

Axis status information is displayed in the upper right portion of the tab.

Dominating the center of the tab is a real-time windstream plot of Following Error and DAC output. Buttons below the plot can temporarily freeze the windstream, turn on or off automatic plot axis scaling, shrink or expand the axis scaling by factors of two, and choose the desired units for displaying Following Error.

If desired, the more common tuning parameters may be inspected and modified using the table near the bottom of the tab. AxCon will only accept new values if the Tuning button is depressed. The current value of each parameter is shown to the right of each field name. To enter a new value, type the value into the field provided then type Enter; the change will take effect immediately so take extra care if making drastic changes. Changing the parameters interactively in this way is an effective way to tune the telescope and immediately see the effect quantitatively on the motion behavior of the telescope.

For the main azimuth and elevation axes, it is recommended to test the performance of a candidate tuning set at both a nominal sidereal rate of 15 arc seconds per second and a nominal slew rate of a few degrees per second and note the maximum following error. It is also recommended to test simultaneous azimuth and elevation slewing to check possible moment coupling.

Changes made to the tuning parameters by AxCon are not permanent; default values will again be installed whenever the system is restarted. To make the changes permanent you must edit the TelescopeServer.dmx file ServoSystem structure (§4.2.2.1.7). Note that the Reset button within AxCon restores the values saved in the PMAC firmware, *not* the values that may be installed via the TelescopeServer.dmx file.

AxCon displays messages and errors across the bottom of the main window and appends them to a separate scrolled text window for later review. This separate window may be iconified if desired but will always be reposted again if any new entries are added.

4.4 HIGH-LEVEL TOOLS

These tools provide a fairly high level view of the telescope system. They require the core framework processes to be running.

4.4.1 DEVICE BROWSER

This tool provides a direct connection to each of the core framework server processes. The browser is generic in the sense that it queries each of the core servers for their legal commands and provides a GUI to access each of those commands. It knows nothing about what the commands actually do.

In the upper left pane is a list of each server and its status. Clicking on one of these will display two panes on the right. The upper right pane shows each of the read-only Attributes and their current values. The lower right pane provides a field or button to activate each of the Commands. Clicking Execute brings up a dialog with the specific fields for that command. Set the fields as desired and click Execute to send the command.

4.4.2 STARCAL

This tool provides a means to perform, evaluate and perhaps install a pointing model. A pointing model forms the conversion between astronomical coordinates and the encoders on the telescope mount. The idea is to point to several stars at known positions and record the encoder values at the moment the star is centered, then compute the weights of several arithmetic and trigonometric terms to form the best statistical fit between the stars and the encoder values. See the StarCal document for full details.

4.4.3 TELESCOPE INTERFACE

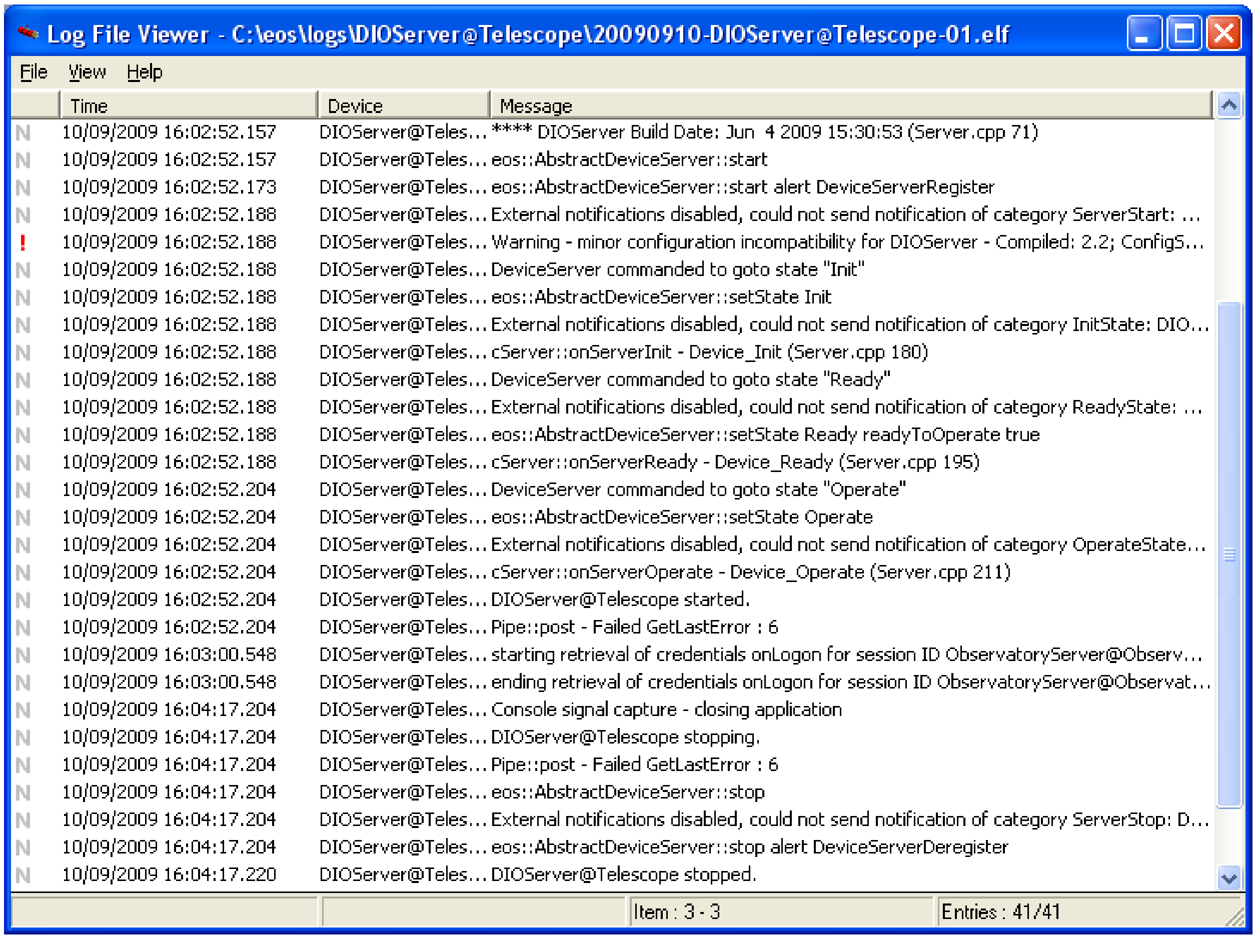
This is a convenient yet comprehensive GUI for operating each of the basic functions of the telescope. The functionality is divided up among several tabs, any one of which is made visible at a time by selecting it from the left column. See the Telescope Interface document for more details.

CHAPTER 5 LOG VIEWER

Logview is a desktop application used to display system messages contained in log files written by Framework processes. These files are in a compact binary format so they can not be viewed with an ordinary text editor. Check these log files whenever more information is required about a telescope event.

Start Log Viewer and use **File→Open** to select a log file for viewing. The log files are in c:\eos\logs. Log files are named with the device that created them and have the extension .elf.

After opening a log file, you will see a window similar to the following:


Log File Viewer - C:\eos\logs\DIOServer@Telescope\20090910-DIOServer@Telescope-01.elf
File
View
Help

```
    Time                      Device           Message
N   10/09/2009 16:02:52.157   DIOServer@Teles… **** DIOServer Build Date: Jun  4 2009 15:30:53 (Server.cpp 71)
N   10/09/2009 16:02:52.157   DIOServer@Teles… eos::AbstractDeviceServer::start
N   10/09/2009 16:02:52.173   DIOServer@Teles… eos::AbstractDeviceServer::start alert DeviceServerRegister
N   10/09/2009 16:02:52.188   DIOServer@Teles… External notifications disabled, could not send notification of category ServerStart: …
!   10/09/2009 16:02:52.188   DIOServer@Teles… Warning - minor configuration incompatibility for DIOServer - Compiled: 2.2; ConfigS…
N   10/09/2009 16:02:52.188   DIOServer@Teles… DeviceServer commanded to goto state "Init"
N   10/09/2009 16:02:52.188   DIOServer@Teles… eos::AbstractDeviceServer::setState Init
N   10/09/2009 16:02:52.188   DIOServer@Teles… External notifications disabled, could not send notification of category InitState: DIO…
N   10/09/2009 16:02:52.188   DIOServer@Teles… cServer::onServerInit - Device_Init (Server.cpp 180)
N   10/09/2009 16:02:52.188   DIOServer@Teles… DeviceServer commanded to goto state "Ready"
N   10/09/2009 16:02:52.188   DIOServer@Teles… External notifications disabled, could not send notification of category ReadyState: …
N   10/09/2009 16:02:52.188   DIOServer@Teles… eos::AbstractDeviceServer::setState Ready readyToOperate true
N   10/09/2009 16:02:52.188   DIOServer@Teles… cServer::onServerReady - Device_Ready (Server.cpp 195)
N   10/09/2009 16:02:52.204   DIOServer@Teles… DeviceServer commanded to goto state "Operate"
N   10/09/2009 16:02:52.204   DIOServer@Teles… eos::AbstractDeviceServer::setState Operate
N   10/09/2009 16:02:52.204   DIOServer@Teles… External notifications disabled, could not send notification of category OperateState…
N   10/09/2009 16:02:52.204   DIOServer@Teles… cServer::onServerOperate - Device_Operate (Server.cpp 211)
N   10/09/2009 16:02:52.204   DIOServer@Teles… DIOServer@Telescope started.
N   10/09/2009 16:02:52.204   DIOServer@Teles… Pipe::post - Failed GetLastError : 6
N   10/09/2009 16:03:00.548   DIOServer@Teles… starting retrieval of credentials onLogon for session ID ObservatoryServer@Observ…
N   10/09/2009 16:03:00.548   DIOServer@Teles… ending retrieval of credentials onLogon for session ID ObservatoryServer@Observat…
N   10/09/2009 16:04:17.204   DIOServer@Teles… Console signal capture - closing application
N   10/09/2009 16:04:17.204   DIOServer@Teles… DIOServer@Telescope stopping.
N   10/09/2009 16:04:17.204   DIOServer@Teles… Pipe::post - Failed GetLastError : 6
N   10/09/2009 16:04:17.204   DIOServer@Teles… eos::AbstractDeviceServer::stop
N   10/09/2009 16:04:17.204   DIOServer@Teles… External notifications disabled, could not send notification of category ServerStop: D…
N   10/09/2009 16:04:17.204   DIOServer@Teles… eos::AbstractDeviceServer::stop alert DeviceServerDeregister
N   10/09/2009 16:04:17.220   DIOServer@Teles… DIOServer@Telescope stopped.

                                  Item : 3 - 3          Entries : 41/41
```

Figure 2 Typical Log File Viewer window

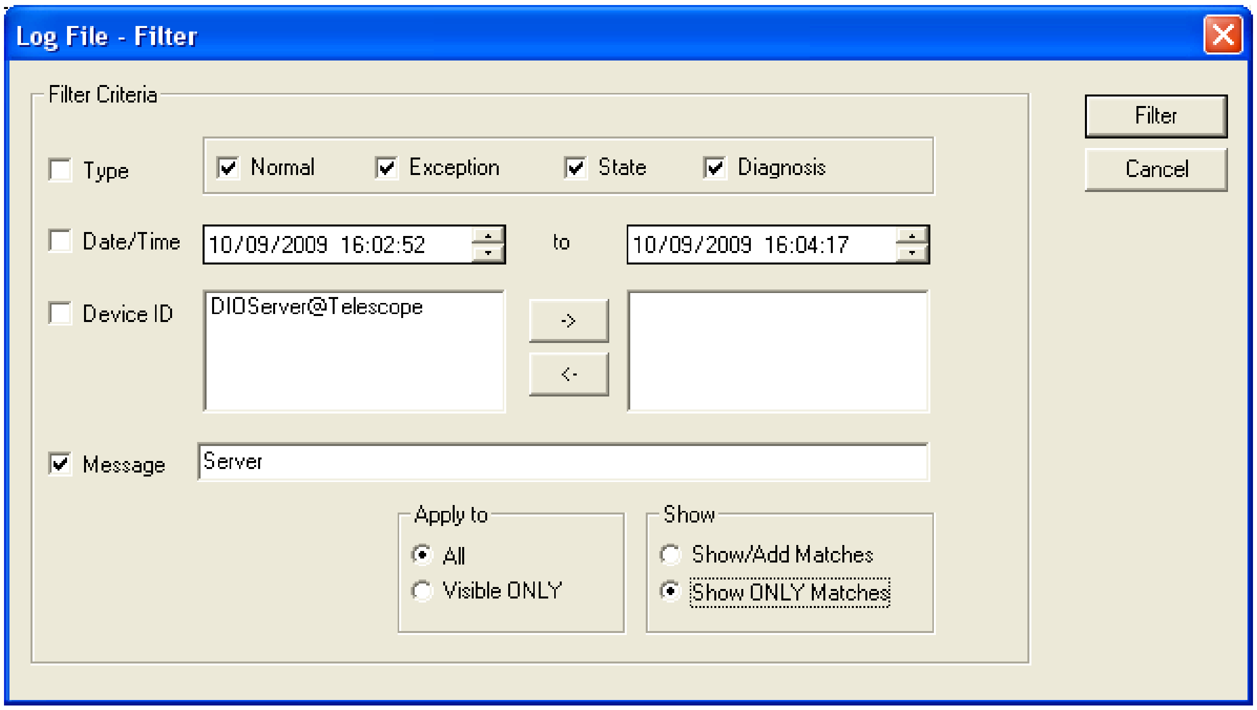
There is one log entry per line, sorted by increasing time. The first column is an indication of whether it is a normal, exception, state or diagnosis **Category**. The second column is the date and **Time** at which the entry was created, in UTC. The third column is the name of the Framework **Device** that created the entry. The last column is the log **Message** text.

The window can be resized and the individual column widths can be adjusted as desired. If an entry does not fit within the available window width, either resize the window larger or move the cursor over it and it will be displayed temporarily in its entirety in a floating bubble window.

At any one time, one entry is the current entry, indicated by being shown white on black. Clicking on a line will make it the current entry, or it can be moved up or down by using the arrows on the keyboard.

5.1 FILTERS

The messages can be restricted to only those that match a set of filter conditions. To use the filter, open **View→Filter**. This opens a dialog similar to the following:


Log File - Filter

Filter Criteria

☐ Type — ☑ Normal ☑ Exception ☑ State ☑ Diagnosis

☐ Date/Time — 10/09/2009 16:02:52 to 10/09/2009 16:04:17

☐ Device ID — DIOServer@Telescope

☑ Message — Server

Apply to: [✓ All] Visible ONLY

Show: Show/Add Matches [✓ Show ONLY Matches]

Figure 3 Log Viewer Filter dialog

The set of checkboxes down the left side determine which of several possible selection criteria are to be applied. Any or all may be used, as desired. The **Type** allows filtering by message type. **Date/Time** allows restricting to a range of dates. The **Device ID** can be one or more of those found within the open log file. The **Message** criteria can be used to select only those log entries that contain the given string exactly.

The **Apply to** box allows using the filter system to successively refine a search in multiple steps. If the choice **Visible ONLY** is used, then the next **Filter** operation only narrows down the entries that are currently already shown in the main window. If the **All** choice is used, then it effectively starts over from the full set of log entries.

After setting the choices as desired, click **Filter** to apply the choices.

After applying a series of filters, the entire log can be viewed again by using **View→Clear All Filters.**

A slight variation of the Filter idea is used by **View→Find**. This will advance through the log file to highlight the next entry that matches the given set of criteria. To look for the next matching entry, just type **F3**.

5.2 OTHER CONTROLS

Log Viewer only reads the selected file only once when it is **Open**ed. If it has changed since it was opened, use **View→Refresh** to read it again.

When studying a large log file, it can be helpful to set **Bookmarks**. These can be controlled using the appropriate controls in the **View** menu. A bookmark at the current line can be **Toggle**d on or off, and the screen can be scrolled to show the **Next** or **Previous** bookmark. These functions can be performed from the keyboard using F2, in combination with the shift and control keys, as shown to the right of their respective menu entry.

5.3 VIEWING ON COMMAND LINE

The log files created by the Framework system are stored in a binary format and can not be inspected with an ordinary text application. However, EOSLog2Ascii is a command line tool for converting EOS binary log format (ELF) log files into ASCII format.

To use EOSLog2Ascii type: EOSLog2Ascii, followed by the filename of the elf file you are trying to convert. For example,

```
EOSLog2Ascii 20091018-LogServer-0.elf
```

This will print the contents of the log file to standard output. Use the command line redirect option ('>') to specify the name of a text file to write the contents to if desired. The ASCII output includes the local computer timestamp for the entry, followed by tab-delimited message type, device identification, and log message originated by the device. Timestamp is in the format YYYY/MM/DD HH:mm:ss.sss.

CHAPTER 6 REGULAR MAINTENANCE

This chapter summarizes the software tasks that will require attention from time to time.

6.1 EARTH ROTATION

For maximum pointing and tracking accuracy, the Earth rotation file should be updated occasionally. The file is published weekly and it is recommended it be downloaded to the TCC at least on a monthly basis. The active file is always C:\EOST\DATA\ERP.dat.

To update the Earth rotation file proceed as follows:

1. copy the existing file C:\EOST\Data\ERP.dat somewhere for posterity
2. download http://maia.usno.navy.mil/ser7/mark3.out
3. copy mark3.out to C:\EOST\Data\ERP.dat

No further action is necessary, the telescope server will use the new file the next time it is required for ephemeris calculations.

The file data will be extrapolated if the current MJD is outside the range of the ERP file.

6.1.1 ERP FILE FORMAT

The ERP file format is assumed to be a series of records, one per line, containing the following fields:

1. Prediction flag : 'P' if predicted, blank if measured (ignored)
2. Year : long (ignored)
3. Month : long (ignored)
4. Day : long (ignored)
5. Modified Julian Date : float
6. X Pole : CEP-IRP seconds of arc north, float
7. X Pole error : float (ignored)
8. Y Pole : CEP-IRP seconds of arc west, float
9. Y Pole error : float (ignored)
10. UT1-UTC : seconds of time, float
11. UT1-UTC error : float (ignored)

Leading records that do not meet this format are ignored. Fields are separated by one or more spaces, and there may be extra spaces at the beginning or end of each line. The lines may end in a carriage-return/line-feed pair or just with a line-feed (Unix-style format). Records must be in time order and are normally at 1-day intervals, although this is not mandatory.

For more information on these data see http://hpiers.obspm.fr/iers/bul/bulb/explanatory.html.

APPENDIX A: COMMON PMAC COMMANDS

PMAC Cheat Sheet

#n	address axis n
J+	jog positive @ Ixx22
J-	jog negative @ Ixx22
J/	jog to stop
J={p}	jog to encoder position p
J:{c}	jog by c encoder counts from current commanded position
J^{c}	jog by c encoder counts from current actual position
O{p}	run open loop at power level p% of Ixx69
P	report position
V	report velocity
F	report following error
$$$	global reset from EEPROM
$$$***	global reset to factory defaults
SAVE	save everything to EEPROM
k	kill current axis
<ctrl>k	kill all axes
<ctrl>d	stop all PLCs
Enable plc1	restart *all* PLCs
Ixx19	jog acc, counts/ms^2
Ixx22	jog vel, counts/ms
Ixx24	$800000 + $02000 to disable both limits
	$800000 + $10000 to ignore amp fault
Ixx69	max DAC output – O is % of this
Ixx30	K_P proportional gain
Ixx31	K_D derivitive gain
Ixx32	K_{VFF} velocity feed-forward gain
Ixx33	K_I integral gain
Ixx35	K_{AFF} acceleration feed-forward gain
Ixx34	0 = integrate all the time, 1 = integrate only when commanded vel=0
Ixx63	Integral windup limit
Ixx68	friction feedforward
Mxx14	enable amp *if* PLCs *not* running
Mxx21	report + end-of-travel limit
Mxx22	report – end-of-travel limit
Mxx02	DAC output
Mxx01	raw encoder value, just 24 bits
Mxx03	latched encoder value

Stylized servo equation for DAC output:

$$DAC = K_P\ (E_F + K_{VFF}\ C_V + K_{AFF}\ C_A + K_I\ \Sigma E_F - K_D\ A_V\ )$$

where:

E_F is following error
C_V and C_A are commanded velocity and acceleration
A_V is actual velocity.

APPENDIX B: AXCON CONFIG FILE

```
# Sample AxCon config file

0_MotName     Az (\#9)               # motor common name, appears as tab string
0_Enc1        0, Raw Azimuth A       # first encoder number, common name
0_Enc2        1, Raw Azimuth B       # second encoder number, common name (if required)
0_PosUnits    degrees                # name for units displayed as string
0_PosScale    1182222.222222         # enc counts per PosUnits
0_InitialPos  0                      # initial position, PosUnits
0_MaxVel      4                      # max vel, PosUnits/s
0_MaxAcc      1                      # max acc, PosUnits/s2

1_MotName     El (\#13)
1_Enc1        2, Raw Elevation A
1_Enc2        3, Raw Elevation B
1_PosUnits    degrees
1_PosScale    1182222.222222
1_InitialPos  0
1_MaxVel 2
1_MaxAcc 1
```

APPENDIX C: SCHEMA-DIOSERVER.DMS

For pedagogical use only — may not exactly match final version installed in system.

```
/* Schema-DIOServer.dms                                              */
/* Message schema file for the DIOServer                             */

/* Project: Lick APF 2.4m Telescope */


__Schema_Version__
{
    __Schema_Version_Major__2 INTEGER
    __Schema_Version_Minor__0 INTEGER
}

EventInfo
{
    Reason              STRING      /* Reason for the fault state */
    Date                STRING      /* UTC Date-Time the fault occurred */
}

ApplicationInfoStruct
{
    Version             STRING      /* Application version Information */
    BuildInfo           STRING      /* Application build information */
    SchemaVersionMajor  INTEGER     /* Schema major version */
    SchemaVersionMinor  INTEGER     /* Schema minor version */
    Simulated           BOOLEAN     /* Server is running in simulate mode */
    Trace               BOOLEAN     /* Server is running in tracing mode */
    Fault               STRUCTURE EventInfo     /* Current fault, if any */
}

HardwareInfoStruct
{
    Number              INTEGER     /* Number of the board in the computer: 1-based on Windows
system, 0-based on Linux */
    Type                STRING      /* Type of board */
    DriverVersion       STRING      /* Driver version number string */
}

TelescopeStatusStruct
{
    ComputerCmdEnableStatus            BOOLEAN
    EnableDomeCommandsStatus           BOOLEAN
}

TelescopeFaultStruct
{
    EmergencyStopStatus                BOOLEAN
    TelescopeEmergencyStopStatus       BOOLEAN
    DomeEmergencyStopStatus            BOOLEAN
    EmergencyCloseStatus               BOOLEAN
    ComputerEmergencyCloseStatus       BOOLEAN
    DomeEmergencyCloseStatus           BOOLEAN
    FaultStatus                        BOOLEAN
    FailsafeFaultStatus                BOOLEAN
    FailsafeFaultAzPosStatus           BOOLEAN
    FailsafeFaultAzNegStatus           BOOLEAN
    FailsafeFaultElPosStatus           BOOLEAN
    FailsafeFaultElNegStatus           BOOLEAN
    FailsafeFaultSecAPosStatus        BOOLEAN
    FailsafeFaultSecANegStatus        BOOLEAN
    FailsafeFaultSecBPosStatus        BOOLEAN
    FailsafeFaultSecBNegStatus        BOOLEAN
    FailsafeFaultSecCPosStatus        BOOLEAN
    FailsafeFaultSecCNegStatus        BOOLEAN
    FailsafeFaultSecATipTiltStatus    BOOLEAN
    FailsafeFaultSecBTipTiltStatus    BOOLEAN
    FailsafeFaultSecCTipTiltStatus    BOOLEAN
    FailsafeFaultPMACWatchdogStatus   BOOLEAN
    FailsafeOverrideStatus            BOOLEAN
    EncoderFaultStatus                BOOLEAN
    FuseFaultStatus                   BOOLEAN
    PowerSupplyFaultStatus            BOOLEAN
    PowerSupplyENCPFaultStatus        BOOLEAN
    PowerSupply24VFaultStatus         BOOLEAN
    PowerSupplyHVAFaultStatus         BOOLEAN
    PowerSupplyHVBFaultStatus         BOOLEAN
    DomeCWLimitSwStatus               BOOLEAN
    DomeCCWLimitSwStatus              BOOLEAN
    DomeAzStopCmdStatus               BOOLEAN
}

MirrorCoverStruct
{
    MCOpenStatus                      BOOLEAN
    MCClosedStatus                    BOOLEAN
    MCFailsafeStatus                  BOOLEAN
    MCPaddleOpenCmdStatus             BOOLEAN
    MCPaddleCloseCmdStatus            BOOLEAN
    OpenMCEnableStatus                BOOLEAN
    CloseMCEnableStatus               BOOLEAN
    MCAOpenSwStatus                   BOOLEAN
    MCAClosedSwStatus                 BOOLEAN
    MCBOpenSwStatus                   BOOLEAN
    MCBClosedSwStatus                 BOOLEAN
    MCCOpenSwStatus                   BOOLEAN
    MCCClosedSwStatus                 BOOLEAN
    MCDOpenSwStatus                   BOOLEAN
    MCDClosedSwStatus                 BOOLEAN
}

DeviceServer
{
    Attributes STRUCTURE
    {
        ApplicationInfo  STRUCTURE ApplicationInfoStruct
        HardwareInfo     STRUCTURE HardwareInfoStruct
    }
    Commands STRUCTURE
    {
        GetApplicationInfo STRUCTURE
        {
            Parameters STRUCTURE
            {
            }
            Returns STRUCTURE
            {
                ApplicationInfo  STRUCTURE ApplicationInfoStruct
            }
        }
        GetHardwareInfo STRUCTURE
        {
            Parameters STRUCTURE
            {
            }
            Returns STRUCTURE
            {
                HardwareInfo  STRUCTURE HardwareInfoStruct
            }
        }
        Reset STRUCTURE
        {
            Parameters STRUCTURE
            {
            }
            Returns STRUCTURE
            {
            }
        }
        DebugTrace STRUCTURE
        {
            Parameters STRUCTURE
            {
                Trace  BOOLEAN
            }
            Returns STRUCTURE
            {
            }
        }
        DebugSetFault STRUCTURE
        {
            Parameters STRUCTURE
            {
            }
            Returns STRUCTURE
            {
            }
        }
    }
    Devices STRUCTURE
    {
        TelescopeStatus STRUCTURE
        {
            Attributes STRUCTURE
            {
                Status  STRUCTURE TelescopeStatusStruct
            }
            Commands STRUCTURE
            {
                GetStatus STRUCTURE
                {
                    Parameters STRUCTURE
                    {
                    }
                    Returns STRUCTURE
                    {
                        Status  STRUCTURE TelescopeStatusStruct
                    }
                }
                EnableCommands STRUCTURE
                {
                    Parameters STRUCTURE
                    {
                    }
                    Returns STRUCTURE
                    {
                    }
                }
                DisableCommands STRUCTURE
                {
                    Parameters STRUCTURE
                    {
                    }
                    Returns STRUCTURE
                    {
                    }
                }
            }
        }
        TelescopeFaults STRUCTURE
        {
            Attributes STRUCTURE
            {
                Status  STRUCTURE TelescopeFaultStruct
            }
            Commands STRUCTURE
            {
                GetStatus STRUCTURE
                {
                    Parameters STRUCTURE
                    {
                    }
                    Returns STRUCTURE
                    {
                        Status  STRUCTURE TelescopeFaultStruct
                    }
                }
                EmergencyClose STRUCTURE
                {
                    Parameters STRUCTURE
                    {
                    }
                    Returns STRUCTURE
                    {
                    }
                }
                EmergencyCloseCancel STRUCTURE
                {
                    Parameters STRUCTURE
                    {
                    }
                    Returns STRUCTURE
                    {
                    }
                }
            }
        }
        MirrorCover STRUCTURE
        {
            Attributes STRUCTURE
            {
                Status  STRUCTURE MirrorCoverStruct
            }
            Commands STRUCTURE
            {
                GetStatus STRUCTURE
                {
                    Parameters STRUCTURE
                    {
                    }
                    Returns STRUCTURE
                    {
                        Status  STRUCTURE MirrorCoverStruct
                    }
                }
                OpenMirrorCovers STRUCTURE
                {
                    Parameters STRUCTURE
                    {
                    }
                    Returns STRUCTURE
                    {
                    }
                }
                CloseMirrorCovers STRUCTURE
                {
                    Parameters STRUCTURE
                    {
                    }
                    Returns STRUCTURE
                    {
                    }
                }
                StopMirrorCovers STRUCTURE
                {
                    Parameters STRUCTURE
                    {
                    }
                    Returns STRUCTURE
                    {
                    }
                }
            }
        }
    }
}
```

APPENDIX D: DIOSERVER.DMX

For pedagogical use only — may not exactly match final version installed in system.

```
/* DIOServer.dmx                                                */
/* Network configuration file for the DIOServer                 */

/* Project: Lick APF 2.4m Telescope */


__Schema_Version__
{
    __Schema_Version_Major__2 INTEGER
    __Schema_Version_Minor__2 INTEGER
}

HardwareConfig
{
    Number          INTEGER         /* Number of board in computer, 0 based */
    Type            STRING          /* Type of board, Currently support types: (ni) pci-dio-96,
and (mc) pci-dio96 */
}

TriggerStruct
{
    If              ARRAY INTEGER   /* List of digital inputs that must be satisfied to cause a
trigger */
}

InputConfig
{
    Name            STRING          /* A human-readable name for the bit */
    Number          INTEGER         /* The index of the bit in the dio port to pin maping */
    Invert          BOOLEAN         /* True inverts the signal, false reports it as read */
    SimulatedInput  BOOLEAN         /* The input value to report for this bit when the server is
simulated, default = false */
    LogPeriod       FLOAT           /* Logging period for signal changes in seconds, omit to
disable */
    FilterPeriod    FLOAT           /* Filtering time constant for signal changes in seconds,
omit to disable */
}

OutputConfig
{
    Name            STRING          /* A human-readable name for the command */
    SetTrue         ARRAY INTEGER   /* A list of the bits in the dio port to pin mapping that
will get set to true */
    SetFalse        ARRAY INTEGER   /* A list of the bits in the dio port to pin mapping that
will get set to false */
    LogPeriod       FLOAT           /* Logging period for command in seconds, omit to disable */
    Triggers        ARRAY STRUCTURE TriggerStruct
}

DeviceConfig
{
    Name            STRING          /* The name of the matching device in the server's schema */
    FieldName       STRING          /* The name of the matching structure in the server's schema
*/
    Inputs          ARRAY STRUCTURE InputConfig
    Outputs         ARRAY STRUCTURE OutputConfig
```

```
}

ApplicationConfig
{
    Simulate        BOOLEAN         /* Are sensors and devices simulated? */
    Trace           BOOLEAN         /* Is diagnostic tracing enabled? */
    ScanRate        FLOAT           /* Duty cycle rate in seconds */
}

Configuration
{
    Application     STRUCTURE ApplicationConfig
    Hardware        STRUCTURE HardwareConfig
    Devices         ARRAY STRUCTURE DeviceConfig
}


@

/* ComputerBus lines:
   DIO Card = Measurement Computing PCI-DIO-96 board
   ComputerBusIn00 = line 48
   ComputerBusIn01 = line 50
   ComputerBusIn02 = line 52
   ComputerBusIn03 = line 54
   ComputerBusIn04 = line 88
   ComputerBusIn05 = line 90
   ComputerBusIn06 = line 92
   ComputerBusIn07 = line 94

   ComputerBusOut00 = line 00
   ComputerBusOut01 = line 01
   ...
   ComputerBusOut46 = line 46
   ComputerBusOut47 = line 47
   ComputerBusOut48 = line 72
   ComputerBusOut49 = line 74
   ComputerBusOut50 = line 76
   ComputerBusOut51 = line 78
   ComputerBusOut52 = line 80
   ComputerBusOut53 = line 82
   ComputerBusOut54 = line 84
   ComputerBusOut55 = line 86
*/

/* Compatible with DIOP Interface Version 5 */

Configuration
{
    Application
    {
        Simulate  FALSE
        Trace     FALSE
        ScanRate  1.0
    }
    Hardware
    {
        Number    0
        Type      PCI-DIO96
    }
    Devices
    [
        {
            Name       "TelescopeStatus"
            FieldName  "TelescopeStatusStruct"
            Inputs
            [
                {
                    Number          33      /* ComputerBusOut33 */
                    Name            ComputerCmdEnableStatus
                    SimulatedInput  TRUE
                }
                {
                    Number          37      /* ComputerBusOut37 */
                    Name            EnableDomeCommandsStatus
                    SimulatedInput  TRUE
                }
            ]
            Outputs
            [
                {
                    /* Special case: Initalize is called upon Server startup */
                    Name            Initialize
                    SetTrue         [48]    /* ComputerBusIn00 */
                    /* Note: this line is active-low */
                }
                {
                    Name            EnableCommands
                    SetFalse        [48]    /* ComputerBusIn00 */
                    /* Note: this line is active-low */
                }
                {
                    Name            DisableCommands
                    SetTrue         [48]    /* ComputerBusIn00 */
                    /* Note: this line is active-low */
                }
            ]
        }
        {
            Name "TelescopeFaults"
            FieldName "TelescopeFaultStruct"
            Inputs
            [
                {
                    Number           0      /* ComputerBusOut00 */
                    Name            EmergencyStopStatus
                }
                {
                    Number           1      /* ComputerBusOut01 */
                    Name            TelescopeEmergencyStopStatus
                }
                {
                    Number           2      /* ComputerBusOut02 */
                    Name            DomeEmergencyStopStatus
                }
                {
                    Number           3      /* ComputerBusOut03 */
                    Name            EmergencyCloseStatus
                }
                {
                    Number           4      /* ComputerBusOut04 */
                    Name            ComputerEmergencyCloseStatus
                }
                {
                    Number           5      /* ComputerBusOut05 */
                    Name            DomeEmergencyCloseStatus
                }
                {
                    Number           6      /* ComputerBusOut06 */
                    Name            FaultStatus
                }
                {
```

```
        Number          7      /* ComputerBusOut07 */
        Name           FailsafeFaultStatus
    }
    {
        Number          8      /* ComputerBusOut08 */
        Name           FailsafeFaultAzPosStatus
    }
    {
        Number          9      /* ComputerBusOut09 */
        Name           FailsafeFaultAzNegStatus
    }
    {
        Number         10      /* ComputerBusOut10 */
        Name           FailsafeFaultElPosStatus
    }
    {
        Number         11      /* ComputerBusOut11 */
        Name           FailsafeFaultElNegStatus
    }
    {
        Number         12      /* ComputerBusOut12 */
        Name           FailsafeFaultSecAPosStatus
    }
    {
        Number         13      /* ComputerBusOut13 */
        Name           FailsafeFaultSecANegStatus
    }
    {
        Number         14      /* ComputerBusOut14 */
        Name           FailsafeFaultSecBPosStatus
    }
    {
        Number         15      /* ComputerBusOut15 */
        Name           FailsafeFaultSecBNegStatus
    }
    {
        Number         16      /* ComputerBusOut16 */
        Name           FailsafeFaultSecCPosStatus
    }
    {
        Number         17      /* ComputerBusOut17 */
        Name           FailsafeFaultSecCNegStatus
    }
    {
        Number         18      /* ComputerBusOut18 */
        Name           FailsafeFaultSecATipTiltStatus
    }
    {
        Number         19      /* ComputerBusOut19 */
        Name           FailsafeFaultSecBTipTiltStatus
    }
    {
        Number         20      /* ComputerBusOut20 */
        Name           FailsafeFaultSecCTipTiltStatus
    }
    {
        Number         21      /* ComputerBusOut21 */
        Name           FailsafeFaultPMACWatchdogStatus
    }
    {
        Number         22      /* ComputerBusOut22 */
        Name           FailsafeOverrideStatus
    }
    {
        Number         23      /* ComputerBusOut23 */
        Name           EncoderFaultStatus
```

```
                }
                {
                    Number          24      /* ComputerBusOut24 */
                    Name            FuseFaultStatus
                }
                {
                    Number          25      /* ComputerBusOut25 */
                    Name            PowerSupplyFaultStatus
                }
                {
                    Number          26      /* ComputerBusOut26 */
                    Name            PowerSupplyENCPFaultStatus
                }
                {
                    Number          27      /* ComputerBusOut27 */
                    Name            PowerSupply24VFaultStatus
                }
                {
                    Number          28      /* ComputerBusOut28 */
                    Name            PowerSupplyHVAFaultStatus
                }
                {
                    Number          29      /* ComputerBusOut29 */
                    Name            PowerSupplyHVBFaultStatus
                }
                {
                    Number          34      /* ComputerBusOut34 */
                    Name            DomeCWLimitSwStatus
                }
                {
                    Number          35      /* ComputerBusOut35 */
                    Name            DomeCCWLimitSwStatus
                }
                {
                    Number          36      /* ComputerBusOut36 */
                    Name            DomeAzStopCmdStatus
                }
            ]
            Outputs
            [
                {
                    /* Special case: Initalize is called upon Server startup */
                    Name            Initialize
                    SetFalse        [50]    /* ComputerBusIn01 */
                }
                {
                    Name            EmergencyClose
                    SetTrue         [50]    /* ComputerBusIn01 */
                }
                {
                    Name            EmergencyCloseCancel
                    SetFalse        [50]    /* ComputerBusIn01 */
                    Triggers
                    [
                        /* Cancel if Cmds Disabled */
                        { If [ -33 4 ] }

                        /* Cancel if Fault */
                        { If [ 6 4 ] }
                    ]
                }
            ]
        }
        {
            Name "MirrorCover"
            FieldName "MirrorCoverStruct"
```

```
Inputs
[
    {
        Number          30      /* ComputerBusOut30 */
        Name            MCOpenStatus
        SimulatedInput  TRUE
    }
    {
        Number          31      /* ComputerBusOut31 */
        Name            MCClosedStatus
    }
    {
        Number          32      /* ComputerBusOut32 */
        Name            MCFailsafeStatus
    }
    {
        Number          38      /* ComputerBusOut38 */
        Name            MCPaddleOpenCmdStatus
    }
    {
        Number          39      /* ComputerBusOut39 */
        Name            MCPaddleCloseCmdStatus
    }
    {
        Number          40      /* ComputerBusOut40 */
        Name            OpenMCEnableStatus
    }
    {
        Number          41      /* ComputerBusOut41 */
        Name            CloseMCEnableStatus
    }
    {
        Number          42      /* ComputerBusOut42 */
        Name            MCAOpenSwStatus
        SimulatedInput  TRUE
    }
    {
        Number          43      /* ComputerBusOut43 */
        Name            MCAClosedSwStatus
    }
    {
        Number          44      /* ComputerBusOut44 */
        Name            MCBOpenSwStatus
        SimulatedInput  TRUE
    }
    {
        Number          45      /* ComputerBusOut45 */
        Name            MCBClosedSwStatus
    }
    {
        Number          46      /* ComputerBusOut46 */
        Name            MCCOpenSwStatus
        SimulatedInput  TRUE
    }
    {
        Number          47      /* ComputerBusOut47 */
        Name            MCCClosedSwStatus
    }
    {
        Number          72      /* ComputerBusOut48 */
        Name            MCDOpenSwStatus
        SimulatedInput  TRUE
    }
    {
        Number          74      /* ComputerBusOut49 */
        Name            MCDClosedSwStatus
```

```
                }
            ]
            Outputs
            [
                {
                    /* Special case: Initalize is called upon Server startup */
                    Name                Initialize
                    SetFalse            [52 54] /* ComputerBusIn02 ComputerBusIn03 */
                }
                {
                    Name                OpenMirrorCovers
                    SetTrue             [52]    /* ComputerBusIn02 */
                    SetFalse            [54]    /* ComputerBusIn03 */
                }
                {
                    Name                CloseMirrorCovers
                    SetFalse            [52]    /* ComputerBusIn02 */
                    SetTrue             [54]    /* ComputerBusIn03 */
                }
                {
                    Name                StopMirrorCovers
                    SetFalse            [52 54] /* ComputerBusIn02 ComputerBusIn03 */
                    Triggers
                    [
                        /* Stop opening once opened */
                        { If [ 30 40 ] }

                        /* Stop closing once closed */
                        { If [ 31 41 ] }

                        /* Stop if Cmds Disabled */
                        { If [ -33 40 ] } { If [ -33 41 ] }

                        /* Stop if Fault */
                        { If [ 6 40 ] }  { If [ 6 41 ] }
                    ]
                }
            ]
        }
    ]
}
```

INDEX