AzCam Users Manual

Michael Lesser

University of Arizona

Imaging Technology Laboratory

04 March, 2010

Table of Contents

Introduction

AzCam is a software image data acquisition system developed at the University of Arizona Imaging Technology Laboratory. It currently operates Astronomical Research Cameras, Inc. Gen3, Gen2, and Gen1 CCD controllers and Magellan Guider controllers.

The AzCam web site containing the latest version of this document as well as files for downloading is located at http://azcam.itl.arizona.edu.

This manual assumes AzCam is already installed and configured for a specific system. AzCam installation and configuration is described in another document, AzCamInstallationManual. The **AzCamTool** graphical user interface (GUI) for AzCam in described in AzCamToolManual.

AzCam Commands

This section describes AzCam syntax and commands. It applies to releases 4.0 and later. It is intended for relatively advanced users of AzCam, including those writing scripts and client applications.

The most up to date command information is always available from the automatically generated HTML files linked below. It is useful to have some basic knowledge of python when reading those files.

- This link jumps to the AzCam HTML command description index

Conventions

Commands (or methods) and Parameters (or attributes) are named with MixedCase convention, as in `SetFormat(1,512,1,512,1,1)` or `Version`.

Commands must have parentheses following their names, even if no parameters are required, as in `Reset()`.

Commands and Parameters are case sensitive. While some commands passed into the ControllerServer program are case insensitive, it is best to use the proper case for all commands and parameters.

Filenames should be given with forward slash ('/') separators, even on Windows machines. If back slashes are used for some reason, they must be doubled as in `c:\\data`.

Strings must be enclosed in quotation marks (single preferred), as in `Get('Version')`. Quotation marks must match ("Version' is not acceptable). A quotation mark may be included in a string by preceding it with a backslash ("I am Mike\'s dog.")

AzCam commands should not use Python's print statement, but instead use the `Print()` function, which prints to the AzCamMonitor system. Scripts may use the print statement as they are designed to run from the command line. The issue here is that AzCam commands are often run in threads by remote clients and printing can easily become jumbled.

Versioning

AzCamServer consists of many different modules, some of which may be dynamically loaded, as well as remote controller/telescope/instrument server code. There is therefore no single version number or date which uniquely identifies all the code. The following convention is used:

A “build version” for AzCam can be read with ***Get('Version').*** It is a version string such as '4.45' and is incremented when significant changes are made.

All python source code modules may be examined for version information at the top of each file.

Objects

At its lowest level, AzCam utilizes object-based commands which provide separate control of the various aspects of AzCam functionality. There is unique code contained in the various objects which interacts with hardware such as controllers, instruments, and telescopes and well as within more virtual objects such as the databases, communication interfaces, etc.

The required command syntax is ***object.Command()*** where *object* is the object name (such as controller, instrument, telescope) and ***Command()*** is the command to be sent. If ***Command()*** uses parameters, they are specified as comma separated values of the appropriate type, such as ***object.command('ITL',1.234,45).***

For example, to send the command ***CompsOn()*** to the instrument, use: ***instrument.CompsOn().***

To send the ***GetHeader()*** command to the telescope, use: *telescope.GetHeader().*

A controller command might be: ***controller.SetRoi(1,100,1,200,2,3).***

All object commands return a python list in the format: ***[status,value1,value2,..]*** where status is the string OK or ERROR. OK means the command executed successfully and ERROR means the command encountered a problem. When status is ERROR, then value1 is always an error message string describing the problem. The error message string is enclosed in double quotes, as in: ERROR "bad parameter specified for Action".

When reply to a client, the python list is converted to a space delimited string, so that returning ***['OK',value1,'value2',..]*** would become OK value1 "**value2**”.

User Commands

A high level user command syntax is also provided in additional to object commands. User commands are defined which do not require object names and which internal call the proper lower level object commands. Generally user commands return the same status list as object commands, but there are many user commands which do not for the sake of simplicity.

Attributes

Python attributes (variables or parameters) may be read with the ***Get()*** command and written with the ***Set()*** command. It is recommended that ***Get()*** and ***Set()*** be used when reading and writing attributes from remote clients. For example: ***Get('controller.TimingBoardInstalled')*** returns the value of the ***TimingBoardInstalled*** parameter from the ***controller*** object, as well as its data type, ‘int’ in this case.

Note that ***Set('instrument.SomeParameter')*** is not the same as ***Set('telescope.SomeParameter')***, since *SomeParameter* in each case is an attribute of a different object. If no object is provided, then the *Globals* object is assumed (which contains global variables across all objects).

Database or header/keyword Commands

AzCam uses object specific keyword indexed databases to maintain textual informational about the system. The keywords and their corresponding values, data type, and comment field are stored in each of the controller, instrument, and telescope databases. These databases are manipulated by commands both from clients and internally in AzCam. Many of the database values are written to the image file header (such as a FITS header) when an exposure begins. The databases are accessed through methods such **as *controller.Header.GetAllKeywords()*** and ***instrument.Header.GetKeyword('FILTER1')***.

The ***ReadHeader()*** method of each object will actively read **hardware** to obtain header information (as ***controller.ReadHeader()*** or ***instrument.ReadHeader()***). This is very different from the object.Header methods which only manipulate the internal Header databases.

The telescope and instrument databases are considered temporary and re-read every time an exposure starts. This is so that rapidly changing data values do not become stale.

Most database information is written to the image file header if the selected image format supports headers. When an object such as an instrument or telescope is disabled, the corresponding object database information is deleted and no longer updated.

See header commands for header command details.

Exposure Commands

Expose() is used to take a complete exposure. It does not return until the entire exposure sequence has completed.

Expose1() is the same as ***Expose()*** but returns immediately. The exposure process then runs in a seperate thread so that clients may query for exposure status (integration time and pixel readout). There is a similar command ***Guide1()*** which immediately returns after starting the ***Guide()*** command.

SetRoi(FirstCol=-1,LastCol=-1,FirstRow=-1,LastRow=-1,ColBin=-1,RowBin=-1,RoiNum=0) is used to set the region of interest. The -1 default means to use the current values, so all parameters do not need to be specified. ***RoiNum*** defaults to 0 and so is not needed for single ROI systems. For example, to change binning only, use:

SetRoi(-1,-1,-1,-1,ColBin,RowBin)

GetRoi() may return ***'OK 1 100 1 100'***.

Temperature Commands

GetTemperatures() return all temperatures. The values are obtained from the last time the ***ReadTemperatures()*** command was executed, which is done internally in the background. Typically the values are no more than a few seconds old, but may not be updated at certain times, such as during image readout. The temperatures are in degrees Celsius. Any number of temperature values may be returned in the return list, as available for the specific system.

SetTemperature(Temperature) sets the detector control temperature. For example: ***SetTemperature(-105.0)*** sets the control temperature of the default sensor to -105.0 Celsius.

Globals.TemperatureSource may be set to 'instrument' or 'controller' to define where the temperature sensors are read. This allows the camera controller or a separate temperature controller to be used for temperature control.

Instrument Commands

Note that the base instrument commands are often overridden by specific instrument commands. See documentation for the specific instrument.

Instrument commands are issued as methods and attributes of the instrument object, which must be defined in the SystemModule.

For all commands, the ***InstrumentTypeID*** parameter indicates a specific mechanism when multiple mechanisms are available. There is always a default value for this parameter (which may depend in the instrument) and therefore it does not need to be explicitly listed. An example would be multiple focusing mechanisms in which each mechanism can be controlled separately.

Instrument comparisons are usually lamps used for data calibration. For some systems, comparisons may be unique mechanisms such as Fe-55 X-ray sources or LED illumination panels. Comparisons are specified as a string or a list of strings for multiple values (e.g. 'quartz' or ['quartz','fene','hene'].

Not all instruments support all these commands and additional commands and attributes may be available. See the documentation for specific instrument details.

A specific instrument module (class definition) must be located in the python search path, but is not usually included with the main AzCam distribution. It is common to place the file in SystemFolder.

See instrument commands for instrument command details.

Telescope Commands

Telescope commands are generally not used directly by users or clients, but are used internally in AzCam during an exposure to update the image header and for focus control during focus exposures. The commands are made available to clients for scripting, debugging, and system checkout. Note that it may be possible for clients to communicate directly with the appropriate telescope server rather than indirectly through AzCam.

For all commands, the ***TelescopeTypeID*** parameter indicates a specific mechanism when multiple mechanisms are available. There is always a default value for this parameter (which may depend on the telescope) and therefore it does not need to be explicitly listed. An example would be multiple focusing mechanisms in which each mechanism can be controlled separately.

Not all telescopes support all the commands listed below and additional commands and attributes may be available for some telescopes.

A specific telescope module file must be located in the python search path, it may not be distributed with the main AzCam distribution. It is common to place specific telescope and instrument files in SystemFolder.

See telescope commands for command details.

AzCamMessages and AzCamMonitor

AzCamMessages is a console-like window which by default appears automatically along with the main AzCam window and is used to display messages. These messages do not appear in the main console window since there would then be a complex mixture of messages from command line commands, internal threading commands which run in the back ground, and remote client commands. AzCamMessages is a special usage of AzCamMonitor.

AzCamMonitor is a python client which can be used to display messages from AzCam on a remote machine. It is for monitoring only, no commands are sent to AzCam from AzCamMonitor. To start AzCamMonitor, run ***StartAzCamMonitor.bat*** from the 'local' folder. Edit the command line parameters ***-s ServerName*** and ***-p PortNumber*** as needed. ***ServerName*** is the host name of the machine running AzCam and ***PortNumber*** is the AzCam monitor port on AzCam (usually one greater than the base port, so typically 2403 for the first AzCam process). Prompts will be displayed if no command line parameters are specified.

Scripting

Scripts may be executed within AzCam for data acquisition and image analysis. These scripts must be written in pure python.

There are many files which define the many AzCam commands. These files must be imported into a script before they can be used. For example,

from Scripting import *

is almost always needed for scripts to properly execute. Other imports may include ExposureCommands, ImageCommands, DisplayCommands, or PlotCommands.

After these files are imported, the AzCam commands are available to those scripts, for example, ***Expose()*** or ***Reset().***

The general form for calling the lowest level AzCam commands within scripts is ***Globals.controller.SomeControllerCommand()***, or ***Globals.instrument.SomeInstrumentCommand().***

A script cannot be run outside of AzCam (e.g. from File Explorer) since scripts require global data structures which are only defined within AzCam. We do not recommend executing scripts remotely via a client socket connection since there may be complex interactions between plotting windows, te command line console, and the CommandServer running in the background.

Scripts may be included automatically in the AzCam startup procedure by appending their module name or the name of a module which imports them to the UserCommands list. This is typically done in the SiteConfiguration or SystemModule files. In this case the script command names are available at the command line or to remote clients just like built-in AzCam commands.

Scripts are often executed using the ***Run*** command, as ***Run GetTemps***. This method has the advantage of loading the script each time the run command is called which is useful when debugging a script. In the first method above the script is only imported when AzCam starts and subsequent script edits are not registered. When arguments are supplied on the command line using **Run** they must be space delimited and not placed in parentheses. So ***Run GetTemps 0.2 'logfile1.txt'*** is OK but ***Run GetTemps(0.2,'logfile1.txt')*** is not. Scripts often have an initialization file (.ini) used to read and save defaults values. These files are located in the same folders as the script modules and are usually created automatically as needed. Scripts may also have data files (.txt) (such as lists of exposure times for different filters) which are usually located in the same folder as the script itself.

If a script has a GUI frontend, then the GUI should be called using the script name with a trailing "Gui". As an example, ***Run AcquireImages*** will run the ***AcquireImages.py*** script, while ***Run AcquireImagesGui*** will run the GUI which controls the ***AcquireImages.py*** script. This is just a convention, but it is important so that the ***Run*** command can find the right script and initialization files. One can always Run a script with its absolute pathname, as
Run /data/TestScript.py.

ITL's DetCharScripts

Many example scripts used for detector characterization are available from ITL. A snapshot of the ITL detector characterization scripts can be found at:

- DetCharScripts zip file

See the following link for the auto generated python documentation for the ITL scripts:

http://azcam.itl.arizona.edu/Downloads/DetCharScripts/html/index.html

AzCam DSP Code for ARC and Magellan Controllers

The DSP code which runs in the controllers is assembled and linked with Motorola software tools. The Motorola DSP tools installer is located MotorolaDspTools.msi

This file installs to create to the folder structure ***/AzCam/MotorolaDSPtools/*** which is required by the batch files which assemble and link the DSP source code.

While the AzCam application code for the ARC timing board is typically downloaded during camera initialization, the boot code must be compatible for this to work properly. Therefore AzCam DSP code must be burned into the timing board EEPROMs before use. The AzCam timing DSP code is quite different from the ARC code and is required for AzCam operation. The PCI fiber optic interface board and the utility board use the original ARC code and does not need to be changed. Note this applies to gen3 systems only, the gen1 and gen2 situation is more complex.

For the Magellan systems, there is only one DSP file which must be downloaded during initialization. Note that **.s** files are loaded for the Magellan systems while ***.lod*** files are downloaded for ARC systems.