AzCam User's Manual

Michael Lesser
University of Arizona
Imaging Technology Laboratory
07 Oct, 2011

Table of Contents

Introduction

AzCam is a software image data acquisition and analysis system developed at the University of Arizona Imaging Technology Laboratory. It is currently used for 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.

Additional AzCam documentation includes:

- AzCamInstallationManual containing installation and configuration information
- AzCamToolsManual describing the standard graphical user interface
- HTML documentation of AzCam functions found on the AzCam web site

Versioning

AzCam 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. A major version for AzCam can be read with `Get('Version')`. It is a version string such as '4.60' 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.

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 found on the AzCam web site. It is useful to have some basic knowledge of python when reading those files.

Conventions

Commands (or methods) and attributes (parameters) are named with MixedCase convention, as in `focalplane.SetFormat(1,512,1,512,1,1)` or `CheckAbort()`. Objects (such as `focalplane`) are always all lower case.

Commands must have parentheses following their names even if no attributes are required.

Commands and attributes are case sensitive. While some commands sent to the external ControllerServer program are case insensitive, it is best to use the proper case for all commands and attributes.

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 AzCamLog system. Scripts may use the print statement as they are generally designed to run from the command line. The issue here is that AzCam commands are often run in threads by remote clients and printing to the console can easily become jumbled.

Return Values and Errors

Nearly all AzCam commands return a python list in the format: `[status,value1,value2,..]` where status is the string `OK`, `ERROR`, `WARNING`, or `ABORTED`. `OK` means the command executed successfully. `ERROR` means the command encountered a problem. `WARNING` is a special error case which is considered non-fatal and should not produce a pop-up message in a client application. `ABORTED` means a command or process was aborted, which may or may not be interpreted as an error. Usually warnings appear only in the log window.

When `status` is `ERROR`, then `value1` is always an error message string describing the problem. The error message string is enclosed in quotes, as in:
`ERROR 'bad parameter specified for action'`.

When replying to a client, the python list is converted into a space delimited string, so that a return value of `['OK',value1,'value2',..]` would become `OK value1 'value2'`.

There are a (very) few commands which do not return a status for simplicity, such as `Print()` and some analysis commands which simply return calculated values.

Objects

Python is an object oriented programming language and objects are used extensively in AzCam. Object-based commands provide control of all aspects of AzCam. These commands (*methods*) interact with hardware such as controllers, instruments, temperature controllers, and telescopes as well as with more virtual objects such as the exposures, images, databases, time, 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 attributes, 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 `GetFocus()` command to the telescope, use: `telescope.GetFocus()`.

A `focalplane` command might be: `focalplane.SetRoi(1,100,1,200,2,3)`.

User Commands

There are built-in AzCam commands which are always available to the command line. All standard python commands are also always available to the command line.

A high level "user command" syntax may *optionally* be provided in additional to the standard AzCam commands. User commands are defined on a system dependent basis and may include both object based and commands which do not require object names.

Scripts

Scripts are Python commands which are intended to be run at the AzCam command line. They may or may not be defined as User Commands. Care should be taken when scripts are executed by remote clients since they often require user input (prompting) which is not supported remotely. Scripts are especially useful to invoke GUIs such as PyQt programs and to execute user written commands. Scripts are executed by the `Run` command and must be in the Python search path. See the Scripting section later in this manual.

Attributes

Python attributes 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('ImageType')` returns the current image type. The return value might be ['OK', 'zero', 'str']. The first value being the status, the second ImageType, the third indicating the return values type (useful for clients which see all values as strings.

Object attributes may also be read and changed directly with `Set()` and `Get()`, although clients should be very carefully when modifying objects directly. For example: `Get('controller.TimingBoardInstalled')` returns the value of the `TimingBoardInstalled` parameter from the `controller` object, as well as its data type specified as a string, '`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).

Header Commands

AzCam uses object specific keyword indexed dictionary to maintain textual informational about some objects. These are typically called headers as they are used to provide information in image headers. The keywords and their corresponding values, data type, and comment field are stored in each of the controller, instrument, and telescope `.header` dictionary. These dictionaries are manipulated by commands both from clients and internally in AzCam. Most of the values are written to the image file header (such as a FITS header) when an exposure begins. The dictionaries 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 information (such as `controller.ReadHeader()` or `instrument.ReadHeader()`). This is very different from the `object.Header.xxx` methods which only manipulate the internal Header databases.

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

Most dictionary 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.

AzCamLog and AzCamMonitor

AzCamLog 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. AzCamLog is a special usage of AzCamMonitor.

AzCamMonitor is a python client which can be used to display messages from AzCam in a remote process. It is for monitoring only, no commands are sent to AzCam from AzCamMonitor. To start AzCamMonitor, execute the `StartAzCamMonitor.bat batch file`. 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 any user-defined tasks including data acquisition and image analysis. Scripts must be written in pure python.

There are many modules which define the various AzCam commands. These modules must be imported into a script before they can be used. Scripts should include the line:

```
from AzCam import *
```

to include the base AzCam commands. Scripts should also include the line:

```
from UserCommands import *
```

to include user defined commands, if they exist.

After these files are imported, the AzCam commands are available as for example, `exposure.Expose()` or `controller.Reset()`.

A script cannot be run outside of AzCam (e.g. from File Explorer) since scripts require global data structures which are only defined within the AzCam python enviroment. We do not recommend executing scripts from a remote client since there may be complex interactions between plotting windows, the command line console, and task running in the background. Running some scripts which do not have user interaction can be safe when called from a remote client.

Scripts are usually 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 may have an initialization file (scriptname.ini) which is used to read and save defaults values. These files are located in a subfolder named *params* of the *SystemFolder* and are usually created automatically as needed.

Scripts may be included automatically in the AzCam startup procedure by adding them to the UserCommands module. They can then be executed without the `Run` command (e.g. `SetDefauts()` rather than `Run SetDefaults`).

AzCam DSP Code

The DSP code which runs in the ARC and Magellan controllers is assembled and linked with Motorola software tools. The Motorola DSP tools installer can be found on the AzCam web site.

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 *xxx.s* files are loaded for the Magellan systems while *xxx.lod* files are loaded for ARC systems.