ACE Connector

Software Reference Manual

Created By:
Astronomical Consultants & Equipment, Inc.
2901 W. Sahuaro Divide
Tucson AZ, 85742
U.S.A.

www.astronomical.com

Contents

List of Figures

1 INTRODUCTION

This manual describes installation, setup, and operation of the Connector control system from Astronomical Consultants & Equipment, Inc. (ACE), which allows you to control your telescope and instrumentation.

To use this manual, you need to have the computer(s) you will be using for your control system ready, with operating systems and associated software such as antivirus installed, and your telescope hardware wired and connected. This manual does not cover hardware or wiring of your telescope. These topics are covered in other ACE manuals. You will need to have a basic knowledge of the operating system(s) you will be using. If you will be using the Python bindings, you will need a knowledge of Python programming. Background information like this is beyond the scope of this manual, but you can find information in books and on the Web to help you.

2 Installation and Setup

2.1 Installing Dependencies for ACE Connector

In addition to the core system provided by ACE, Connector depends on additional softare provided by third parties. In the case of Debian systems, these are part of the Debian package system. Parts of Connector may also use IRAF, provided by the National Optical Astronomy Observatory, which can be used to analyze images after they are acquired.

2.1.1 Debian Dependencies

ACE Connector depends on the following packages in Debian:

- libboost1.49.0-dev
- libqtcore4
- libqtgui4
- libqwt6

Begin the installation process by installing these packages using the command:

```
sudo apt-get install <package>
```

for each of the packages listed.

2.1.2 IRAF

The IRAF package is software for astronomical image processing provided by the National Optical Astronomy Observatories. IRAF is not required for ACE Connector systems, but it is useful for image analysis and recommended for use with ACE Connector camera machines. To install IRAF on a Debian machine, follow these instructions.

1. Download IRAF and X11IRAF from http://iraf.noao.edu. As of March 22, 2012, the latest version is 2.16. You will need either the Linux 32-bit or Linux 64-bit compile, depending on your system. You'll also want a copy of X11IRAF. The pre-built binary of X11IRAF for generic Linux is not linked from the front page, but you can find it by going to http://iraf.noao.edu/x11iraf/.

   Make sure you know where the files are saved. In this example, we'll suppose the downloaded files are at

   ```
   /home/tno/Downloads/iraf.lnux.x86.tar.gz and
   /home/tno/Downloads/x11iraf-v2.0BETA-bin.linux.tar.gz
   ```

2. Install tcsh.

```
$ sudo apt-get install tcsh
```

3. If you are using a 64-bit system, you will need to install some 32-bit libraries. If you are using a 32-bit system, you can skip this step.

```
$ sudo dpkg --add-architecture i386
$ sudo apt-get update
$ sudo apt-get install ia32-libs:i386
```

   This step may take a while.

4. Create an account called iraf. You'll need to give it a password.

```
$ sudo adduser iraf
Adding user 'iraf' ...
Adding new group 'iraf' (1002) ...
Adding new user 'iraf' (1002) with group 'iraf' ...
Creating home directory '/home/iraf'
Copying files from '/etc/skel' ...
Enter new UNIX password: <type the password here>
Retype new UNIX password: <retype the same password>
Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] y
```

5. Give the iraf user sudo privilege.

```
$ sudo usermod -a -G sudo iraf
```

6. Change the iraf user's shell to tcsh.

```
$ sudo chsh iraf
Changing the login shell for iraf
Enter the new value, or press ENTER for the default
        Login Shell [/bin/bash]: /bin/tcsh
```

7. Create a directory for the IRAF files.

```
$ sudo mkdir /iraf
```

8. Give ownership of the `/iraf` directory to the iraf user.

```
$ sudo chown iraf /iraf
```

9. Switch to the iraf user's account

   ```
   $ su - iraf
   Password: <type the password you created earlier>
   ```

10. Unpack the downloaded files.

    ```
    $ mkdir /iraf/iraf
    $ mkdir /iraf/x11iraf
    $ cd /iraf/iraf
    $ tar -xzf /home/tno/Downloads/iraf.lnux.x86.tar.gz
    $ cd /iraf/x11iraf
    $ tar -xzf /home/tno/Downloads/x11iraf-v2.0BETA-bin.redhat.tar.gz
    ```

11. Run the IRAF install script. First you'll need to set the iraf environment variable so that the script knows where to look for the files.

    ```
    $ setenv iraf /iraf/iraf
    $ cd /iraf/iraf/unix/hlib
    $ source irafuser.csh
    $ sudo ./install
    ```

    You will be prompted for the iraf user's password.

    When asked whether to proceed to post-installation, say yes.

    When asked whether to configure IRAF networking, and whether to create a default tapecap file, say no.

    When asked whether to delete unused HSI binaries, say yes.

    When asked whether to strip the system of sources, say yes.

    At the end, you should see a message reading:

    ```
    Installation Completed With No Errors
    ```

12. Run the X11IRAF install script.

    ```
    $ cd /iraf/x11iraf
    $ sudo ./install
    ```

    Specify `/usr/local/bin` for the commands directory.

    When asked whether to remove any pre-existing files, say no.

    Accept the defaults on all other questions.

13. Return to your usual user account

```
$ exit
$ whoami
tno
```

14. Create a special directory for working within IRAF.

```
$ mkdir ~/iraf
$ cd ~/iraf
```

15. Use the `mkiraf` command to set up IRAF's working environment.

    IRAF works best when you use xgterm as your terminal.

```
$ mkiraf
-- creating a new uparm directory
Terminal types: xgterm,xterm,gterm,vt640,vt100,etc.
Enter terminal type: xgterm
A new LOGIN.CL file has been created in the current directory.
You may wish to review and edit this file to change the defaults.
```

16. Start xgterm. A new terminal window will appear.

```
$ xgterm &
```

17. In the xgterm window, start IRAF.

```
$ ecl
```

    You should see the following:

```
   NOAO/IRAF PC-IRAF Revision 2.16 EXPORT Thu May 24 15:41:17 MST 2012
      This is the EXPORT version of IRAF V2.16 supporting PC systems.


  Welcome to IRAF.  To list the available commands, type ? or ??.  To get
  detailed information about a command, type 'help <command>'.  To run  a
  command  or  load  a  package,  type  its name.   Type  'bye' to exit a
  package, or 'logout' to get out  of the CL.    Type 'news' to find  out
  what is new in the version of the system you are using.

  Visit http://iraf.net if you have questions or to report problems.

  The following commands or packages are currently defined:

      dataio.     language.   obsolete.   softools.   vo.
      dbms.       lists.      plot.       system.
```

```
        images.     noao.       proto.      utilities.

    ecl>
```

You should now have a working IRAF installation.

2.1.3 PyRAF

PyRAF is a wrapper that makes it easy to write Python scripts that call IRAF tasks. This is useful for creating image processing callbacks used after ACE Connector takes CCD images. To install PyRAF, type:

```
$ sudo pip install pyraf
```

It will take a few minutes to download and install PyRAF.

2.2 Installing ACE Connector Software

This section discusses the installation of ACE Connector. ACE Connector contains the following components:

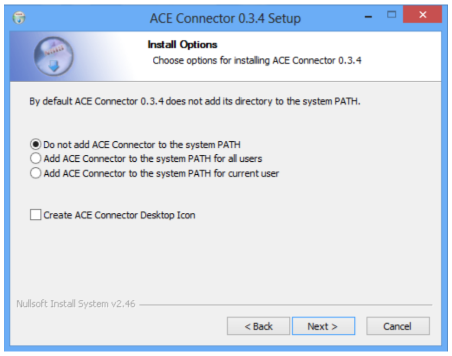
- The core system, which makes it possible for telescope instruments to communicate from one computer to another, and for users to control the instruments.
- Instrument plugins, which allows Connector to control different devices.
- The Graphical User Interface client ("the GUI client"), which provides a user-friendly interface for users to manually control a telescope.
- The Python scripting interface, which allows users to create scripts to automate various telescope operations.

2.2.1 Windows

For Windows machines, an installation package is provided as the file connector-0.4.0-win32.exe. To install ACE Connector, run this program with administrator privileges.

The installer will first ask whether ACE Connector should be added to the system path (Figure 2.1). If you are installing this copy of ACE Connector to interface with the ACE RCS ("Ace.exe"), you need to select either "Add ACE Connector to the system PATH for all users", or "Add ACE Connector to the system PATH for current user". If you aren't sure what to do here, you are probably safe to accept the default.

This part of the installation wizard also gives you the option to create an ACE Connector desktop icon. If you select this, the installer will add a shortcut so that you can launch the client from the desktop.



ACE Connector 0.3.4 Setup

Install Options
Choose options for installing ACE Connector 0.3.4

By default ACE Connector 0.3.4 does not add its directory to the system PATH.

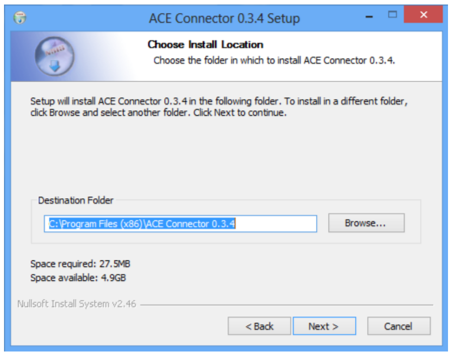
[✓ Do not add ACE Connector to the system PATH]
Add ACE Connector to the system PATH for all users
Add ACE Connector to the system PATH for current user

☐ Create ACE Connector Desktop Icon

Nullsoft Install System v2.46

Figure 2.1: Connector install PATH modification selection

Next, the installer allows you to choose the installation directory (Figure 2.2). If you wish, you may select a custom location. This is useful if you need to re-install Windows in the future. Leaving the default should be fine for most purposes.



ACE Connector 0.3.4 Setup

Choose Install Location
Choose the folder in which to install ACE Connector 0.3.4.

Setup will install ACE Connector 0.3.4 in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

Destination Folder
C:\Program Files (x86)\ACE Connector 0.3.4

Space required: 27.5MB
Space available: 4.9GB

Nullsoft Install System v2.46

Figure 2.2: Connector install directory selection

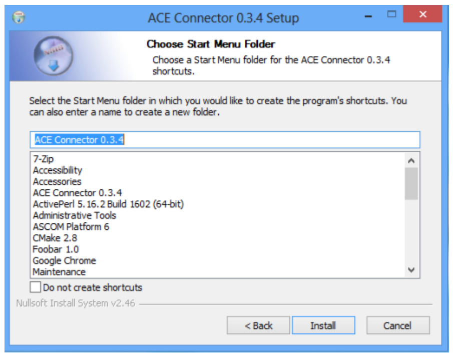
In the next step, you are asked where in the Start Menu you want to place the ACE Connector shortcuts (Figure 2.3). You can also choose not to add shortcuts to the Start Menu by selecting "Do not create shortcuts". Once you select the "Install" button, the installer begins the installation process.

While the installation proceeds, you may receive this error message if you are installing over a pre-existing installation:

> Error opening file for writing:
>
> C:\Program Files (x86)\ACE Connector 0.4.0\bin\acenodesvc.exe
>
> Click Abort to stop the installation, Retry to try again, or Ignore to skip this file.

This error occurs because the pre-existing service is still running when the installation tool tries to overwrite the file. Simply wait for about 30 seconds while the service shuts down, and then select "Retry".

When the installer is finished (Figure 2.4), you should see the following message:



ACE Connector 0.3.4 Setup

Choose Start Menu Folder

Choose a Start Menu folder for the ACE Connector 0.3.4 shortcuts.

Select the Start Menu folder in which you would like to create the program's shortcuts. You can also enter a name to create a new folder.

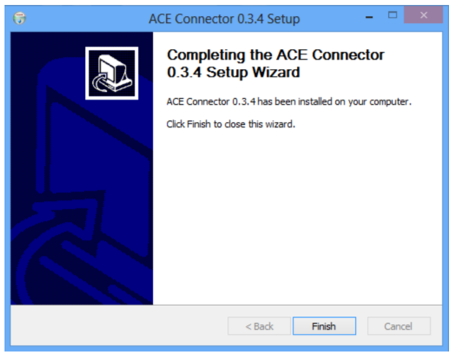
ACE Connector 0.3.4

7-Zip
Accessibility
Accessories
ACE Connector 0.3.4
ActivePerl 5.16.2 Build 1602 (64-bit)
Administrative Tools
ASCOM Platform 6
CMake 2.8
Foobar 1.0
Google Chrome
Maintenance

☐ Do not create shortcuts

Nullsoft Install System v2.46

Figure 2.3: Start menu configuration



ACE Connector 0.3.4 Setup

Completing the ACE Connector 0.3.4 Setup Wizard

ACE Connector 0.3.4 has been installed on your computer.

Click Finish to close this wizard.

Figure 2.4: Windows installation finish screen

ACE Connector 0.4.0 has been installed on your computer.

Click Finish to close this wizard.

ACE Connector is now installed and ready for configuration.

2.2.2 Linux

For Linux installations, a Debian install package is provided. The install package is named connector-0.4.0.deb. To install this package, move to the directory containing the installation package and type the following on the command line:

```
$ sudo dpkg -i connector-0.3.4.deb
```

ACE Connector is now installed and ready for configuration.

2.3 Instrument Control

2.3.1 Adding an Instrument

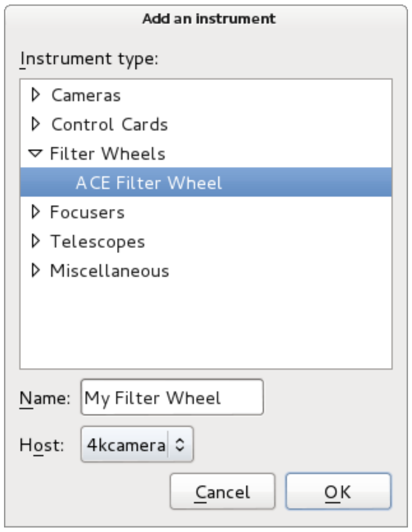


Figure 2.5: The add instrument window

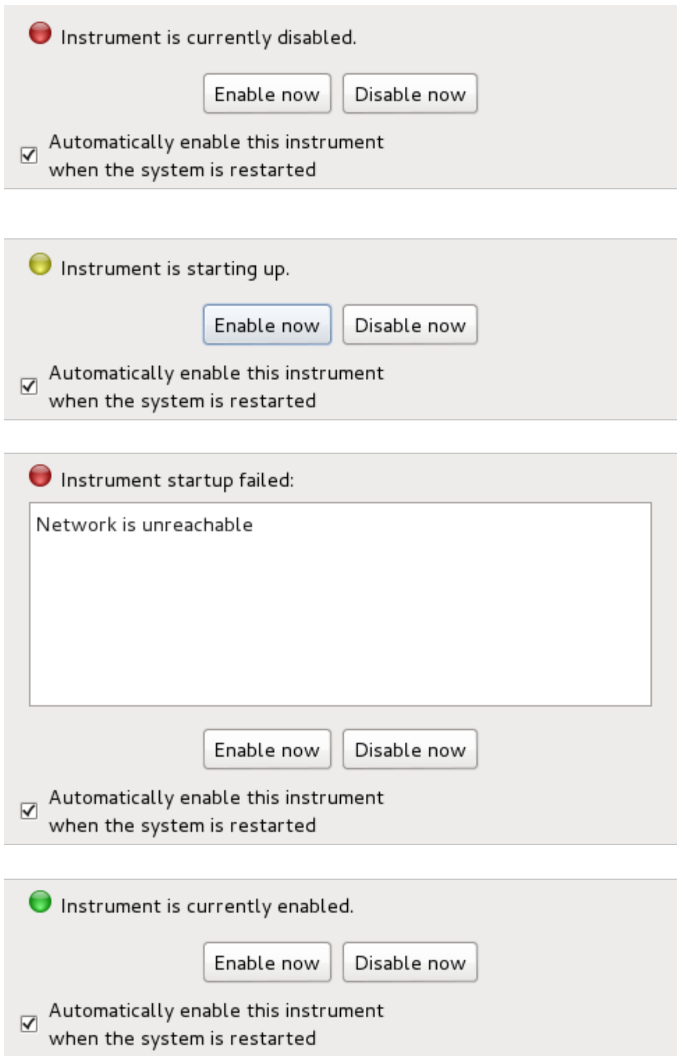
To use telescope hardware with the ACE Connector system, you must first add an instrument to the configuration. Once an instrument is added to your site configuration, you can configure and use it.

To add an instrument, open the configuration window by selecting the screwdriver and wrench icon in the perspectives window. The add instrument window (Figure *The add instrument window*) will appear.

First, select an instrument type to add. In the figure, an ACE Filter Wheel is being added. Next, give the new instrument a unique name. In the example, the instrument is named “My Filter Wheel”. Finally, select a host for the instrument. The host is where the instrument is controlled. Usually the host for an instrument must be the machine to which that instrument is connected.

Select the OK button when you are ready to add the instrument. The instrument's name will appear in the left side of the configuration window. You can now select the instrument's name in the configuration window to change the configuration for the instrument.

2.3.2 Enabling and Disabling Instruments



● Instrument is currently disabled.

☑ Automatically enable this instrument when the system is restarted

● Instrument is starting up.

☑ Automatically enable this instrument when the system is restarted

● Instrument startup failed:

Network is unreachable

☑ Automatically enable this instrument when the system is restarted

● Instrument is currently enabled.

☑ Automatically enable this instrument when the system is restarted

Figure 2.6: Instrument state indicator examples

Instruments used by ACE Connector can be "enabled" and "disabled." When an instrument is enabled, it may be connected or in-use by the ACE Connector system. Usually instruments that are "enabled" within ACE Connector cannot be used by external programs, such as programs that might be provided by the hardware manufacturer. On the other hand, instruments that are "disabled" may be disconnected and are available to other programs on the system, but cannot be used within ACE Connector.

Every instrument's configuration panel includes controls for enabling and disabling the instrument. You will find these below the instrument-specific configuration items. The controls include an indicator that displays the current state of the instrument, two buttons to enable and disable an instrument, and a check-box to choose whether the instrument should be enabled automatically at system start. If you select one of these buttons, the instrument will be enabled or disabled immediately. *If you are enabling an instrument, any changes you have made to the instrument's configuration will be applied immediately without need to select the OK or Apply buttons.*

When you enable an instrument, you will see the indicator change from red to yellow, and the report line will change from "Instrument is disabled" to "Instrument is starting up." When the startup is complete, the indicator's color will change to green, and the report line will change to "Instrument is currently enabled." If an error was encountered, the report line will read "Instrument startup failed" and an error message will be displayed. This error message may be useful in diagnosing the failure.

When you disable the instrument, the indicator will change from green to yellow and the report line will change to "Instrument is shutting down." When the instrument has finished shutting down, the report line will change to "Instrument is currently disabled" and the indicator will become red.

The checkbox controls whether the instrument will be enabled at system start. If the box is checked, when the ACE Node Service (in Windows) or ACE Node Daemon (in Linux) is restarted, or the system is rebooted, the instrument will be enabled at that time. Otherwise, the instrument will be disabled until it is manually enabled.

2.4 Configuration of Cameras

The following sections describe configuration of the optical axis and FITS headers for a CCD. These sections are applicable to any CCD configured with ACE Connector.

2.4.1 Configuring the Optical Axis

When you select the "Edit Optical Axis..." button in the configuration panel for a camera, the optical axis window (Figure 2.7) will be displayed. You can use this window to configure a CCD camera's orientation on the focal plane, and identify other instruments (telescope, filter wheel, and focuser) in the system to be used in conjunction with this CCD.

Offset and rotation parameters must be determined by observation of stars in an image, and comparison of pointing information from the telescope with pointing seen on the CCD. When these parameters are set correctly, the WCS information placed into an image should approximately match the pointing and orientation of the image itself.

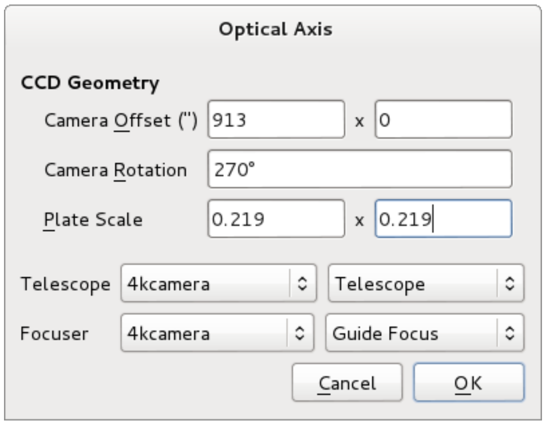
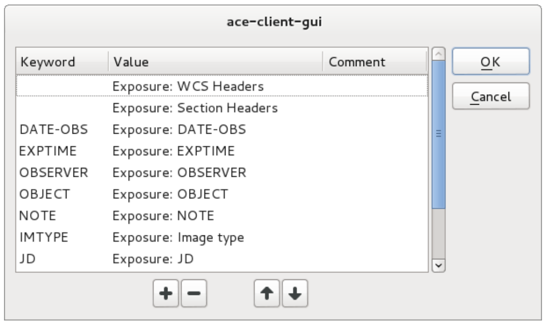

Optical Axis
CCD Geometry
Camera Offset (") 913 x 0
Camera Rotation 270°
Plate Scale 0.219 x 0.219
Telescope 4kcamera Telescope
Focuser 4kcamera Guide Focus
Cancel
OK

Figure 2.7: The Optical Axis Window

2.4.2 Configuring Headers



Keyword	Value	Comment
	Exposure: WCS Headers	
	Exposure: Section Headers	
DATE-OBS	Exposure: DATE-OBS	
EXPTIME	Exposure: EXPTIME	
OBSERVER	Exposure: OBSERVER	
OBJECT	Exposure: OBJECT	
NOTE	Exposure: NOTE	
IMTYPE	Exposure: Image type	
JD	Exposure: JD	

Figure 2.8: The FITS Header Edit Window

When you select the "Edit FITS Header..." button in the configuration panel for a camera, the header edit window (Figure 2.8) will be displayed. This window allows you to select and organize header entries that will appear in images taken by the camera.

To add a new header into the image, select the "plus" button at the bottom of the header edit window. This displays the header card editor window, shown in Figure 2.9.

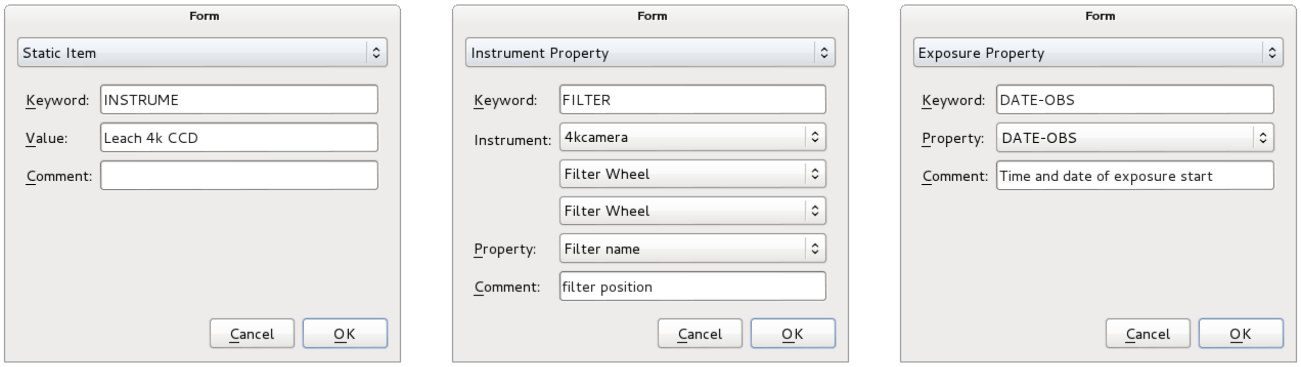
First, you need to select the type of header entry to create from the dropdown box at the top of the header card editor window. There are three different type of header entries that can be added into the system:

Static Item This is a constant string that should be added to the image. Static header items can be used to include header cards like `OBSERVAT` and `INSTRUME` that never change.

To create a static header item, enter the header keyword, the header value, and a comment.

Instrument Property This is a header card that derives from system information. Examples of headers that are instrument properties are:

- Filter wheel position



Static Item

Keyword:	INSTRUME
Value:	Leach 4k CCD
Comment:	

Instrument Property

Keyword:	FILTER
Instrument:	4kcamera
	Filter Wheel
	Filter Wheel
Property:	Filter name
Comment:	filter position

Exposure Property

Keyword:	DATE-OBS
Property:	DATE-OBS
Comment:	Time and date of exposure start

Figure 2.9: Editing three different kinds of FITS header cards

- CCD temperature
- Focus position
- Autoguiding status

To create an instrument property header, supply the keyword for the entry, and select the instrument from the dropdown boxes, the desired property of the instrument, and enter a comment if desired.

Exposure Property This header card supplies properties specific to the exposure. Exposure properties include:

- Object name
- Observer
- Note
- Exposure time
- Image type
- Date of observation (local or universal time)
- Julian date
- Section keywords
- WCS keywords

To create an exposure property header, supply the keyword, select the desired property from the dropdown box, and add a comment if desired. The section and WCS header entries do not require keywords, and the keyword entry will be ignored for such headers.

Existing header cards can be removed from the list by selecting the item in the list of header entries and then selecting the minus-sign button at the bottom of the window. To move an item up or down in the list, use the arrow buttons.

When you are finished editing the image header, select “OK” to accept the changes, or “Cancel” to discard them.

2.5 Camera from Astronomical Research Cameras

Astronomical Research Cameras, Inc., ARC, provide a range of high-end cameras for large telescopes. These cameras are supported by ACE, but a driver is required. The driver may be downloaded from the ARC website.

2.5.1 Linux

The PCI or PCIe card for a Leach Camera is controlled by a Linux driver module provided by ARC. Drivers are provided from ARC for Linux kernel 2.6. If your kernel is newer than this, the driver must be compiled for your kernel version.

To compile the driver, obtain a copy from ACE or from the ARC website. Navigate to the directory `3.5/PCIe/Arc66/2.6.32-358.2.1.el6.i686`.

You will need header files for your kernel. To install kernel header files on a Debian system, type the following command:

```
$ sudo apt-get install linux-headers-$(uname -r)
```

Then type:

```
$ make
```

This should produce a file called Arc66PCIe.ko. To install the file, type:

```
$ sudo mkdir /lib/modules/$(uname -r)/weak-updates
$ sudo mkdir /lib/modules/$(uname -r)/extra
$ sudo ./Load_Arc66PCIe
```

When you install the kernel module, a device list should be printed to the screen. Make sure there is at least one entry in the list.

Your Linux package manager may occasionally (or frequently) upgrade the Linux kernel. Typically you will need to recompile your custom drivers whenever the Linux kernel is updated. This can be burdensome, so you might want to have your package manager put kernel updates on hold. To do this on Ubuntu 12.04 and later, type the following:

```
$ sudo apt-mark hold linux-image-$(uname -r)
```

2.5.2 Windows

To install the Windows driver for the ARC camera, go to http://www.astro-cam.com/arcpage.php?txt=software.php#driver and select the driver appropriate to your Windows version.

Depending on your Windows version, You may need to use test-signed drivers to controll an ARC camera. In Windows Vista and later, Windows must be configured to allow test-mode drivers. To do this, open a command prompt and issue the following command:

```
> bcdedit.exe /set TESTSIGNING ON
```

2.5.3 Configuration

Tip: ARC publishes OWL software for use with its camera. You may use OWL to control the camera independently of ACE Connector, or to help diagnose problems when using ACE Connector with the camera. Make sure that the ARC camera is disabled in the ACE Connector configuration window. Refer to the *Instrument Control* section for more information.

Your camera will be supplied with two DSP files that need to be loaded into the camera firmware each time the camera is initialized. These files are named `tim.lod` and `util.lod`. You will need to place these files somewhere on the computer that hosts the camera's control card. In this example, we'll suppose the files are stored in `/home/tno/system_downloads/`.

You'll also need the following information to fill into the configuration:

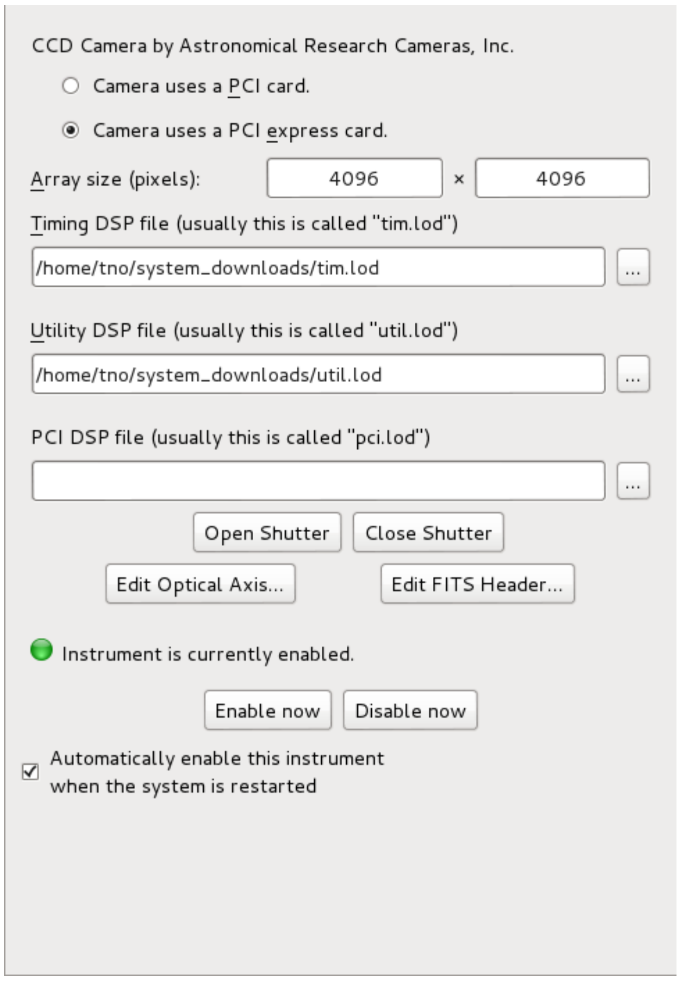
- Whether the control card is PCI or PCIe. If you aren't sure, you probably have a PCIe card. ACE or ARC may be able to verify this information for you.
- The CCD array size in pixels.

If you are setting up the instrument for the first time, check the section on *Adding an Instrument*. You must choose the computer containing the camera's control card as the host for this instrument.

To configure the camera, start the client GUI and open the configuration window by selecting the screwdriver and wrench icon in the Perspectives window, and select your Leach camera from the list in the left pane of the instrument setup window.

The appearance of the configuration panel for the Leach camera is shown in Figure 2.10. Type the names of the DSP files in the relevant field. An example setup is shown. You will need to set up the camera's optical axis and configure the FITS header for the instrument. Refer to the section *Configuration of Cameras* for general information on configuring cameras.

If desired, you may go ahead and enable the camera from this dialog. Select the "Enable now" button to do this. You should see the instrument status change, and the indicator should turn from red to yellow as


CCD Camera by Astronomical Research Cameras, Inc.

- ○ Camera uses a PCI card.
- ● Camera uses a PCI express card.

Array size (pixels): 4096 × 4096

Timing DSP file (usually this is called "tim.lod")

/home/tno/system_downloads/tim.lod

Utility DSP file (usually this is called "util.lod")

/home/tno/system_downloads/util.lod

PCI DSP file (usually this is called "pci.lod")

● Instrument is currently enabled.

☑ Automatically enable this instrument when the system is restarted

Figure 2.10: Configuration panel for an ARC Camera

the camera initializes, followed by green when the initialization is finished. For more information, refer to the section on *Enabling and Disabling Instruments*. If this fails, read the error message and resolve the problem accordingly.

2.5.4 Common Error Messages

The most commonly encountered error messages are presented here, with possible solutions.

```
( CArcPCIe::FindDevices() ): No device bindings exist! Make sure an
Arc, Inc PCIe card is installed!
```

This error message indicates that the control card was not found in the system.

- The driver may not have been installed correctly. You should have a device named `/dev/Arc66PCIe` or similar. You can check that it's there by typing

  ```
  ls -la /dev/Arc66PCI*
  ```

- The card may not be installed in the system. Remember that you must select the computer with the card as the instrument's host when you add it to the ACE Connector configuration.
- Check whether the card is a PCI or PCI Express card. Make sure you have made the correct selection in the configuration panel.

```
( CARCPCIe::ReadReply() ): Time Out [ 2 sec ] while waiting for
status [ 0x41 ]!
```

This error message indicates that the control card is working, but is not able to establish communication with the camera.

- The camera's fiber optic cables may be plugged in incorrectly. For example, the transmit and receive cables may be swapped. Check the fiber optic connections.
- The camera may be powered off. Check power to the camera.

2.6 Camera from Andor Cameras

Important: Only Linux machines may host Andor cameras for ACE Connector at this time.

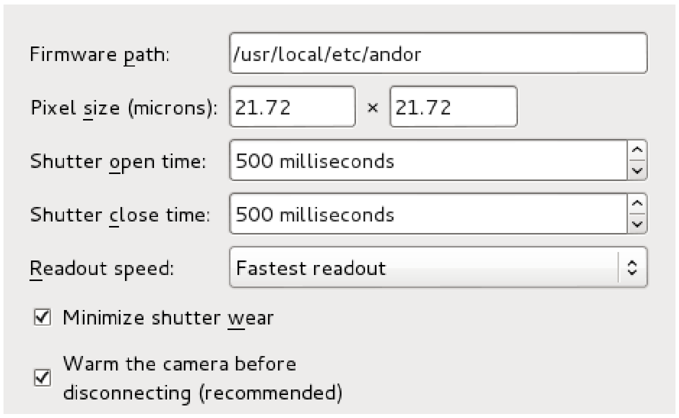
Before ACE Connector can use an Andor camera, the Andor SDK must be installed into the host machine. Obtain a copy of the SDK from Andor and unpack it onto the target system using the command

```
tar -xvf andor.tar.gz
```

This will create a directory structure called `Andor_SDK_2.90.30004.0/SDK/andor` or similar. Inside this directory is a script called `install_andor`. Navigate into the directory and then run the script as root:

```
sudo ./install_andor
```

Type 5 and press enter to install the drivers for your camera model. Once the drivers are installed, you are ready to configure the Andor camera in ACE Connector.



Firmware path: /usr/local/etc/andor
Pixel size (microns): 21.72 × 21.72
Shutter open time: 500 milliseconds
Shutter close time: 500 milliseconds
Readout speed: Fastest readout
☑ Minimize shutter wear
☑ Warm the camera before disconnecting (recommended)

Figure 2.11: Andor configuration panel

To configure a new Andor camera, open the instrument setup window in the client GUI, and select the plus-sign icon at the bottom left corner of the instrument setup window. Select “Andor Camera” from the “Cameras” category. Give the device a name and select the host for the device. The camera must be hosted on the machine to which the camera’s USB cable is connected. Once you add the instrument, you can select the instrument’s name from the list on the left side of the instrument setup window to display the configuration panel shown in Figure 2.11.

The Andor configuration includes the following parameters:

Firmware path The directory containing camera configuration files from Andor. By default these are stored in `/usr/local/etc/andor`.

Pixel size Size of the CCD pixels in microns.

Shutter open time The time required for the shutter to open. The CCD exposure clock does not begin until after the shutter open time has elapsed.

Shutter close time The time required for the shutter to close. At the end of the exposure, the CCD waits this amount of time before beginning CCD readout.

Readout speed The readout setting for the CCD.

Minimize shutter wear If this checkbox is checked, the shutter will not open or close except when necessary. For example, the shutter will not close when an exposure is finished. This minimizes wear on the shutter, which is useful if the camera takes exposures frequently.

Warm the camera before disconnecting If this checkbox is checked, the CCD cooler will slowly raise the temperature when system shutdown is ordered. This reduces thermal stress on the CCD. Selecting this option is recommended. If you select this option, it will take several minutes to disable the camera, so you should do this whenever the camera is not in use.

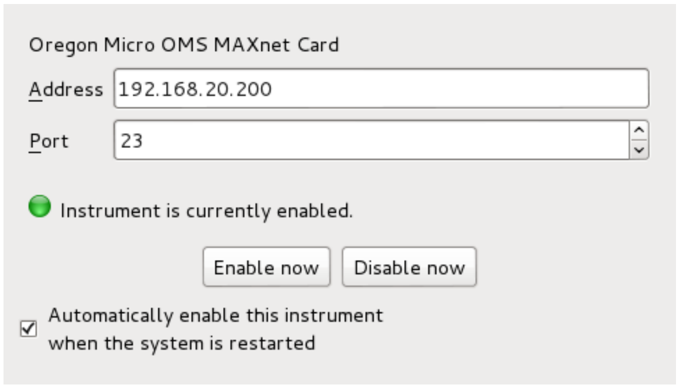
In addition to the above, you should configure your CCD's optical axis and FITS header. Refer to the section *Configuration of Cameras* for further information on configuration of CCD cameras.

2.7 MAXnet Control Card

A MAXnet control card has a network interface that makes available one or more motion control axes as well as general purpose I/O. ACE Connector can use a MAXnet control card to control filter wheels, focusers, and other instrumentation.

To set up a MAXnet card for connector, launch the configuration and add an instrument. For the instrument type, specify "OMS MAXnet Card" under the "Control Cards" category and assign the card a name and a host. The new instrument will now appear in the list on the left side of the instrument configuration window. Select the instrument to finish configuring it.

To use the MAXnet card, you will need to know the card's IP address and port number. The default port for communicating with the MAXnet card is port 23.


Oregon Micro OMS MAXnet Card
Address 192.168.20.200
Port 23
● Instrument is currently enabled.

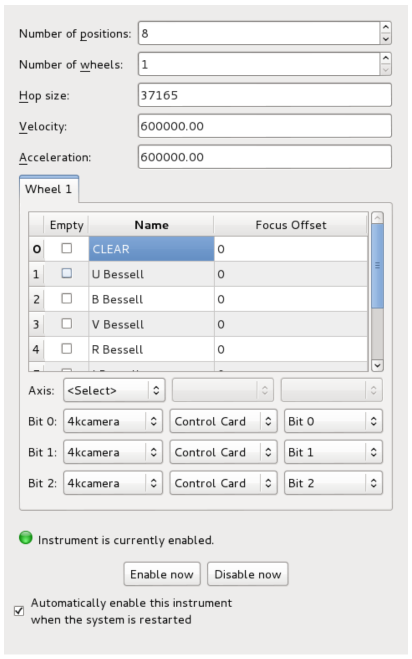
☑ Automatically enable this instrument when the system is restarted

Figure 2.12: MAXnet configuration panel

Type these parameters into the configuration window (Figure 2.12), and check that everything is working by selecting the "Enable now" button. You should see the indicator turn green, and the instrument status should read "Instrument is currently enabled." If the instrument does not initialize correctly, check that you have entered the correct IP address and port number. Verify that you can communicate with the card using a telnet program or with the manufacturer's software.

Once you have configured the MAXnet card, you can configure other instruments to interface with the card.

2.8 ACE Filter Wheel



Number of positions: 8
Number of wheels: 1
Hop size: 37165
Velocity: 600000.00
Acceleration: 600000.00

Wheel 1

	Empty	Name	Focus Offset
0	☐	CLEAR	0
1	☐	U Bessell	0
2	☐	B Bessell	0
3	☐	V Bessell	0
4	☐	R Bessell	0

Axis:	<Select>		
Bit 0:	4kcamera	Control Card	Bit 0
Bit 1:	4kcamera	Control Card	Bit 1
Bit 2:	4kcamera	Control Card	Bit 2

● Instrument is currently enabled.

☑ Automatically enable this instrument when the system is restarted

Figure 2.13: Filter wheel configuration panel

Once you have configured your control card, you can then configure the parameters for a filter wheel using the configuration panel (Figure 2.13). The ACE Filter Wheel comprises one or more wheels, each coupled to a motion axis on a control card as well as I/O bits supplying the position information for the wheel.

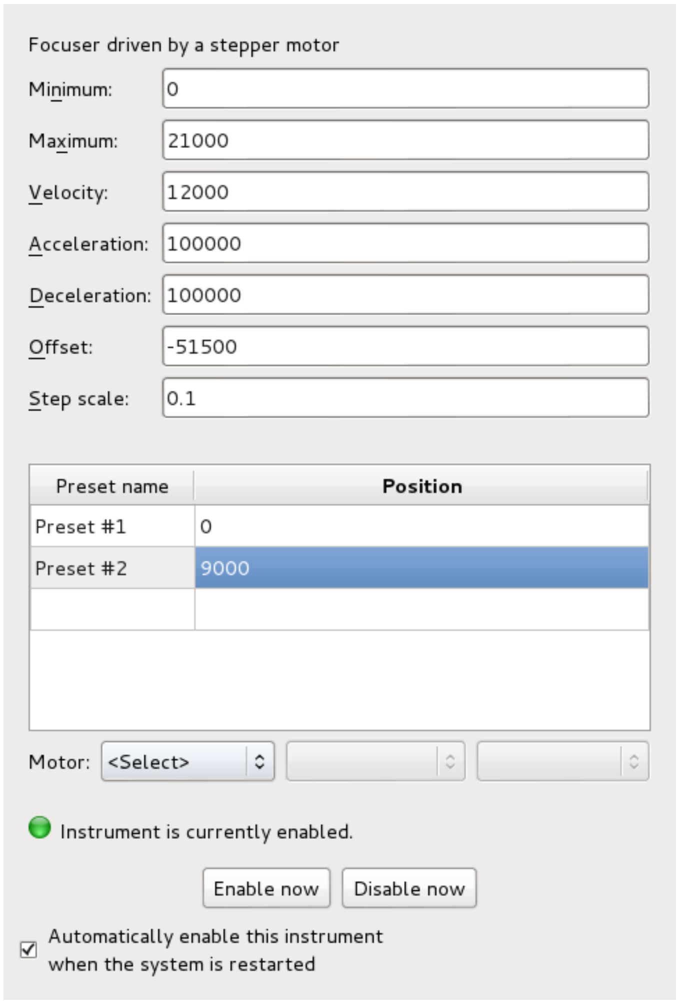
You will need the following information to configure your filter wheel:

- Number of filter positions per wheel
- Number of wheels
- Wheel hop size
- Velocity
- Acceleration
- Motor axis
- Position bit assignments
- Names of the filters in each position
- Focus offset values

The parameters should have been provided to you by ACE. **Do not attempt to use settings other than those given to you by ACE. Doing so could cause your system to work incorrectly and may void your warranty.**

Tip: Because filter wheel names are used in the FITS header, use of only ASCII characters in the names is recommended. IRAF will truncate the filter name at the first space character, so it's best if the first word in the filter name uniquely identifies the filter.

2.9 Focuser Configuration



Focuser driven by a stepper motor

Minimum:	0
Maximum:	21000
Velocity:	12000
Acceleration:	100000
Deceleration:	100000
Offset:	-51500
Step scale:	0.1

Preset name	Position
Preset #1	0
Preset #2	9000

Motor: <Select>

Instrument is currently enabled.

☑ Automatically enable this instrument when the system is restarted

Figure 2.14: Focuser configuration panel

This section describes configuration of focusers that are controlled by a motion axis of a motion control card. Before configuring your focuser, you will need to have configured its motion control card within Connector.

To configure the focuser, you will need the following parameters to type into the configuration panel as shown in Figure 2.14:

Minimum focus position A position accessible but near the reverse limit of the focuser. This will be the minimum position accepted by a focus move command, although the focuser will still be able to move beyond this point through a command to move to the reverse limit.

Maximum focus position The equivalent position for the forward limit.

Velocity The top speed for the focus motor in counts per second.

Acceleration The motor acceleration in counts per second per second.

Deceleration The motor deceleration in counts per second per second.

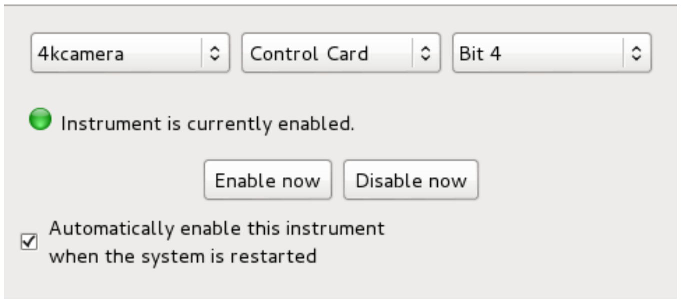
Offset An offset applied between the position reported by the control card axis and that reported by the focuser instrument.

Step scale A scaling factor applied to the position reported by the control card.

Motor axis The control card axis that controls the stepper motor.

The parameters should have been provided to you by ACE. **Do not attempt to use settings other than those given to you by ACE. Doing so could cause your system to work incorrectly and may void your warranty.** You may configure your focuser to have *preset positions*. Once you configure preset positions, buttons appear for each of the presets in the focus control window, and you can access the presets through the Python method `Focuser.go_to_preset()`.

2.10 Switch Configuration


4kcamera | Control Card | Bit 4

● Instrument is currently enabled.

☑ Automatically enable this instrument when the system is restarted

Figure 2.15: Switch device configuration panel

A switch device allows the I/O bit of a control card to be used to switch a mechanical device (a nitrogen valve, lights, etc.) on or off. Before configuring a switch device as described here, you will need to have configured the control card that gives access to the device.

To configure a new switched device, open the instrument setup window in the client GUI, and select the plus-sign icon at the bottom left corner of the instrument setup window. Select "Switched Device" from the "Miscellaneous" category. Give the device a name and select a host for the device. It is recommended that you host the device on the control card's computer.

Once you have added the instrument, you can select the I/O bit that controls the device. Select the new switched device from the list on the left side of the instrument setup window. The device configuration panel (Figure 2.15) will appear on the right side of the window. Use the dropdowns in the configuration panel to select the control card and the I/O bit for the device.

3 Using the Graphical Client

The ACE client GUI is used for day-to-day manual operations of the system provided by ACE Connector.

To start the client GUI in a Windows environment, do one of the following:

1. In a command prompt window, navigate to `C:\Program Files (x86)\ACE Connector\bin` and execute `ace-client-gui.exe`, or
2. From the Start menu, choose "Programs", "ACE Connector", and "ACE Client GUI", or
3. Select the desktop shortcut with the telescope icon.

In Linux, do one of the following:

1. From the command prompt, type `ace-client-gui` and press enter, or
2. Launch the client from your desktop's application launcher. For launchers that sort applications by category, the GUI is usually under the "Science" or "Education" category.

3.1 Using Perspectives

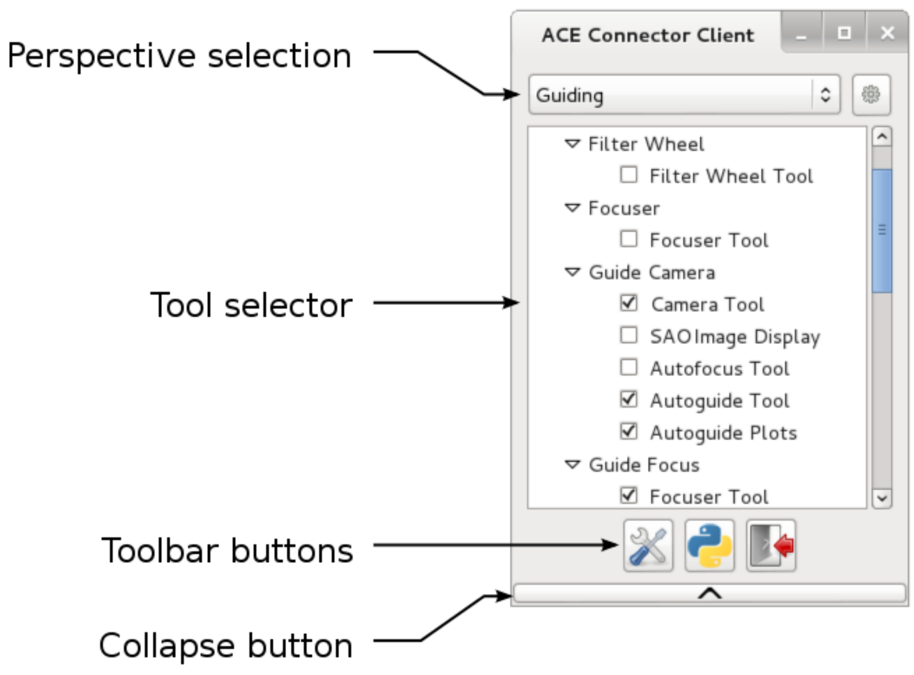
The Connector client GUI is comprised of a number of tool windows that you can use to control instruments with the Connector system. To control which tools are being displayed, you need to use the perspective tool.

When you run the client program for the first time, only the perspective tool (Figure *The perspectives window*) will be displayed. The perspective tool has the following components (from top to bottom):

- The perspective selection box
- The tool selector
- The three toolbar buttons
- The window collapse button

In the tool selector, you will see all of the instruments available through Connector, with all of the tools that can be used to control that instrument. To display a particular tool, select the check box next to the name of that tool. From now on, when you launch the client GUI, the tools you have selected will be displayed. Individual tools are described in the following sections.

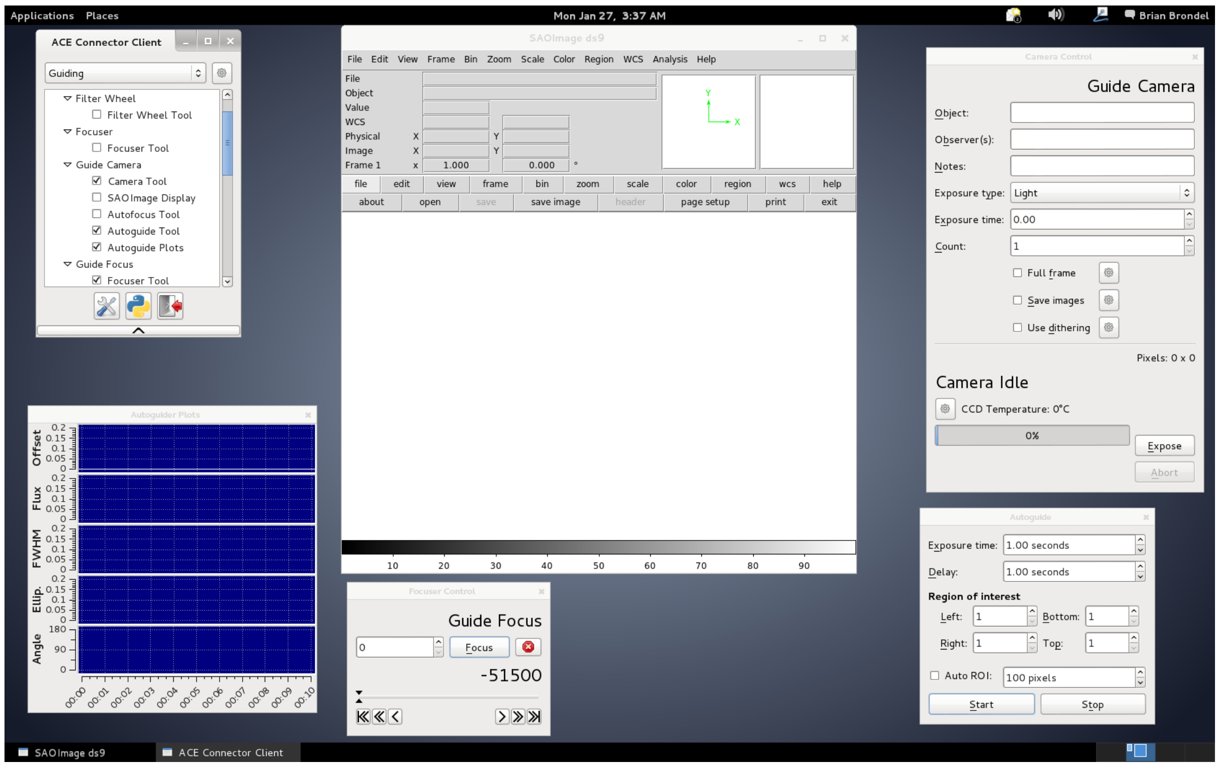
Selecting the collapse button hides the tool selector part of the GUI, allowing the perspective tool to take up less space. Selecting the button again returns the perspective window to the way it was.



Guiding

- Filter Wheel
  - ☐ Filter Wheel Tool
- Focuser
  - ☐ Focuser Tool
- Guide Camera
  - ☑ Camera Tool
  - ☐ SAOImage Display
  - ☐ Autofocus Tool
  - ☑ Autoguide Tool
  - ☑ Autoguide Plots
- Guide Focus
  - ☑ Focuser Tool

Figure 3.1: The perspectives window



Guiding

- Filter Wheel
  - ☐ Filter Wheel Tool
- Focuser
  - ☐ Focuser Tool
- Guide Camera
  - ☑ Camera Tool
  - ☐ SAOImage Display
  - ☐ Autofocus Tool
  - ☑ Autoguide Tool
  - ☑ Autoguide Plots
- Guide Focus
  - ☑ Focuser Tool

Object:
Observer(s):
Notes:
Exposure type: Light
Exposure time: 0.00
Count: 1
☐ Full frame
☐ Save images
☐ Use dithering
Pixels: 0 x 0
CCD Temperature: 0°C
0%

Exposure time: 1.00 seconds
Delay: 1.00 seconds
Region of interest
Left: 1 Bottom: 1
Right: 1 Top: 1
☐ Auto ROI: 100 pixels

Figure 3.2: A desktop set up with tools for using an autoguider

3.1.1 Using Multiple Perspectives

If you wish, you can create multiple different perspectives to use with the client tool. For example, you might create an imaging perspective with tools you need for control of the camera, and an autoguiding perspective with tools for controlling the autoguider. When you switch from one perspective to the other, the tools from one perspective will be removed, and the tools for the new perspective will be displayed.

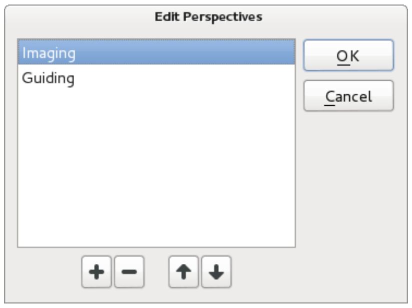


Figure 3.3: The edit perspectives window

To create a new perspective, select the gear button next to the perspective selection box. A window for editing perspectives will appear (Figure 3.3). To create a new perspective, select the plus-sign button. The new perspective will be added to the list with the name “New perspective”. Type in the name for the new perspective to replace the default. If you wish, you can change the order of items in the list with the arrow buttons. When you are finished, select OK. You can now select the new perspective from the perspective selection box in the perspectives window.

3.1.2 The Toolbar

Configuration If you select this button, the configuration window will be opened. This window allows you to edit the configuration of your system.

Python If you select this button, you will be able to select a Python script from your disk drive to run.

Exit Selecting this button will cause the client GUI to exit.

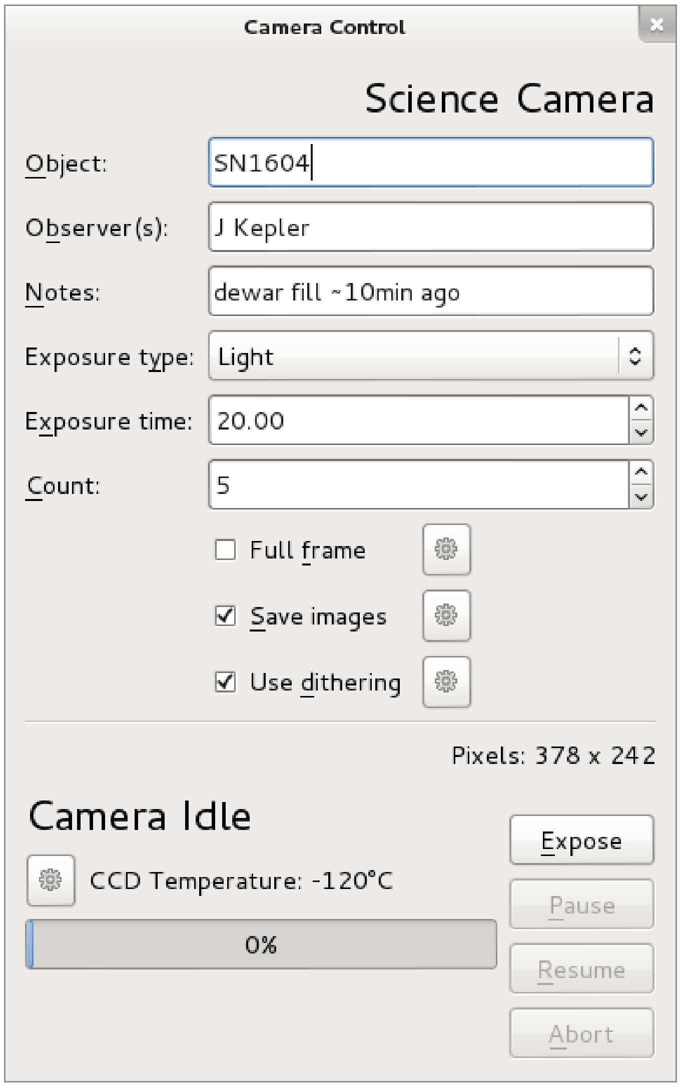
3.2 Controlling a Camera

You can use the camera control tool (Figure 3.4) to control and take exposures with a CCD camera.

Object The object name to store in the image’s FITS header.

Observer(s) The names of the observers to record in the FITS header.

Notes A brief note to associate with the image. This note can be stored in the FITS header.



Object: SN1604

Observer(s): J Kepler

Notes: dewar fill ~10min ago

Exposure type: Light

Exposure time: 20.00

Count: 5

☐ Full frame

☑ Save images

☑ Use dithering

Pixels: 378 x 242

Camera Idle

CCD Temperature: -120°C

0%

Figure 3.4: The camera control tool

Exposure type The type of exposure to take. If "Bias" is selected, an exposure time of zero will be used. If "Dark" is selected, the camera shutter will not be opened during the exposure. Exposure type can (and should) be stored in the FITS header.

Exposure time Desired exposure time in seconds.

Count Number of exposures to take. The system will collect this number of exposures all with the same settings (except that the image sequence number will be incremented).

Full frame If Full frame is checked, the entire unbinned imaging area will be read out into the FITS image, regardless of any region of interest or binning settings. Region of interest, binning, and readout mode of the camera can be configured by selecting the gear icon next to this checkbox. The details of the configuration are described in the following sections.

Save images If Save images is not checked, images will not be saved into the archive or image processing areas on the disk. The images can still be displayed in SAOImage. Parameters for saving and processing images can be configured by selecting the gear icon next to this checkbox. The details of this configuration are described in the following sections.

Use dithering If this is selected, the telescope pointing will be offset by a small random amount before the start of each exposure. The offset is uniformly distributed in a cone centered on the telescope's target position. The maximum dithering radius can be configured by selecting the gear button next to this checkbox.

Note: Information will not be stored into the image header unless the system has been configured to do so. See the section *Configuration of Cameras*.

Only printable ASCII characters (the English alphabet, Arabic numerals, a few commonly used symbols and punctuation marks, and spaces) may be used in FITS headers. If other characters are used in these fields, they may not be recorded correctly in the FITS header.

Buttons at the bottom right corner of the window control the start of the exposure:

- To start the exposure sequence, select the "Expose" button.
- To pause the exposure, select the "Pause" button. The shutter will close, but the CCD will not read out the image.
- When you are ready to continue a paused exposure, select the "Resume" button. The shutter will re-open and the exposure will resume where it left off.
- To abort the exposure sequence without reading it out, press the "Abort" button.

Some cameras do not support the pause function. In this case, the "Pause" and "Resume" buttons will not appear.

3.2.1 Configuring Image Geometry

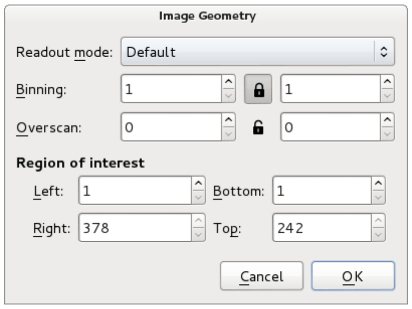


Image Geometry

Readout mode: Default

Binning: 1 | 1

Overscan: 0 | 0

Region of interest

Left: 1 Bottom: 1

Right: 378 Top: 242

Figure 3.5: The image geometry window

If you select the gear button next to the “Full Frame” checkbox in the camera control tool, the image geometry window (Figure 3.5) will be displayed, and you can adjust region of interest, binning, overscan, and readout mode settings for camera exposures.

Readout mode is selected by a combo box at the top of the image geometry window. The available readout modes vary from one camera model to the next.

Images can be unbinned (1x1 binning), or you can read out multiple image pixels at a time in either rows or columns, or both. Usually, images are binned by the same number of pixels in rows and columns. To use different binning in X and Y, de-select the lock icon between the two binning text entries.

Cameras usually don’t support arbitrary binning amounts. Make sure you select binning settings that are supported by your camera. If you choose settings that your camera does not support, the nearest supported setting will be used when exposing the camera.

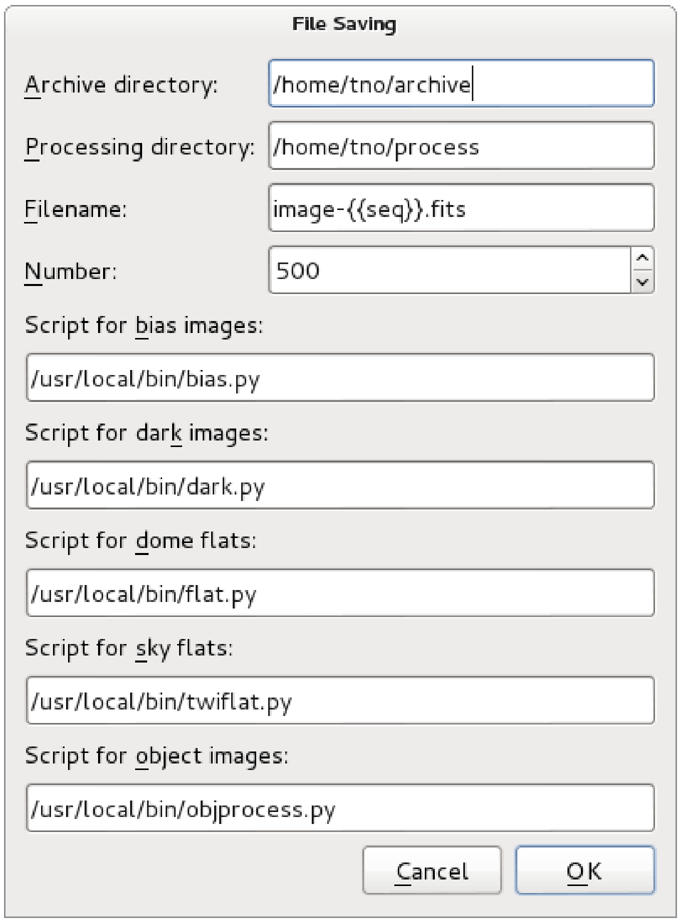
Overscan is an image area read out from the camera after all the pixels in a row have already been read. This provides a measurement of the bias level in the camera. Selecting a non-zero overscan will add that number of pixels to the size of the image X-axis. Binning settings are ignored with respect to this parameter.

Setting a region of interest causes only a section of the image area on a camera to be read out. Specify the unbinned pixel coordinates of the left-most, right-most, top-most, and bottom-most edges of the image. The CCD will read out all the columns between the left and right *inclusive*, and all the rows between the top and bottom *inclusive*.

3.2.2 Image Archiving and Processing

The image saving window (Figure 3.6) controls how CCD images are saved and processed. Parameters that can be specified in this window are:

Archive directory The directory on the camera computer where images will be saved after completion. Images in the archive directory are not modified once they have been collected.


File Saving

Archive directory: /home/tno/archive

Processing directory: /home/tno/process

Filename: image-{{seq}}.fits

Number: 500

Script for bias images: /usr/local/bin/bias.py

Script for dark images: /usr/local/bin/dark.py

Script for dome flats: /usr/local/bin/flat.py

Script for sky flats: /usr/local/bin/twiflat.py

Script for object images: /usr/local/bin/objprocess.py

Figure 3.6: Image saving

Processing directory The directory on the camera computer where images will be processed. If this directory is specified, images will be copied into this directory and the appropriate script will be called, depending on the type of the image.

Filename A template for naming images when they are saved to the archive and processing directories. These templates can include macros such as the image sequence number and filter position. Use of macros is described below.

Number The image sequence number for the next image. This can be used in the filename template.

Script for bias images This can be set to the name of a program or script that will be executed whenever a new bias image becomes available. The program or script is called with the following arguments:

1. The processing directory name as specified above
2. The name of the newly available file

These are also the arguments used in all of the other processing scripts described below.

In addition, a text file is placed into the processing directory called `bias.lis`, which lists all bias images taken since the previous noon.

Script for dark images This can be set to the name of a program or script that will be executed whenever a new dark image becomes available.

In addition, a text file is placed into the processing directory called `dark.lis`, which lists all dark images taken since the previous noon.

Script for dome flats This can be set to the name of a program or script that will be executed whenever a new dome flat becomes available.

In addition, a text file is placed into the processing directory called `flat.lis`, which lists all flat fields taken since the previous noon.

Script for sky flats This can be set to the name of a program or script that will be executed whenever a new sky flat becomes available.

In addition, a text file is placed into the processing directory called `skyflat.lis`, which lists all sky flats taken since the previous noon.

Script for object images This can be set to the name of a program or script that will be executed whenever a new object image becomes available.

In addition, a text file is placed into the processing directory called `object.lis`, which lists all object fields taken since the previous noon.

Note: Image processing scripts are executed with the standard input and standard error streams closed.

The image filename template above can use macros to insert exposure information into the image filename. For example, use of `image-{{filter}}-{{seq}}.fits` for the template will insert the filter name and sequence number into the image filename. Available macros are as follows:

`seq`	Image sequence number. A fixed-width representation can be specified by, e.g., `{{seq:4}}` for a four-digit width
`imtype`	Image type: “light”, “flat”, “skyflat”, “dark”, or “bias”
`filter`	Filter name
`date`	Date of exposure start in the format MM-DD-YYYY
`time`	Time of exposure start in the format HH:MM:SS
`month`	Month of exposure start, 1 = January. Use `{{month:2}}` for zero-padding.
`day`	Day of exposure start. Use `{{day:2}}` for zero-padding
`year`	Year of exposure start. Use `{{year:2}}` for a two-digit year. Otherwise a four-digit year is used.
`hour`	Hour of exposure start. Use `{{hour:2}}` for zero-padding
`min`	Minute of exposure start. Two-digit representation is always used for `{{min}}`.
`sec`	Second of exposure start. Two digit representation is always used for `{{sec}}`.

3.3 The Autoguide Setup Tool

The autoguide tool (Figure 3.7) allows the user to use a CCD camera for telescope guiding. The autoguider uses the following procedure:

1. If the “Auto ROI” feature is selected, take a test image with the requested region of interest and exposure time. As a shorthand, the user may set all of Left, Right, Top, and Bottom to 1 for a full-frame image.
2. Identify the brightest star in the test image.
3. Define a region of interest with the user-specified size, centered on the brightest star in the test image.
4. Take a new image with either the automatically determined region of interest parameters, or the user-specified parameters, depending on whether “Auto ROI” is selected.
5. Find the brightest star in the image. Compare the star’s position to the reference position and calculate a telescope offset.
6. Send a command to apply the offset to the telescope. Wait for the command to complete.
7. Pause for the delay duration specified by the user.
8. Repeat starting with step 4.

The following parameters can be selected when using this tool for autoguiding:

Exposure time The exposure time to use in guide images.

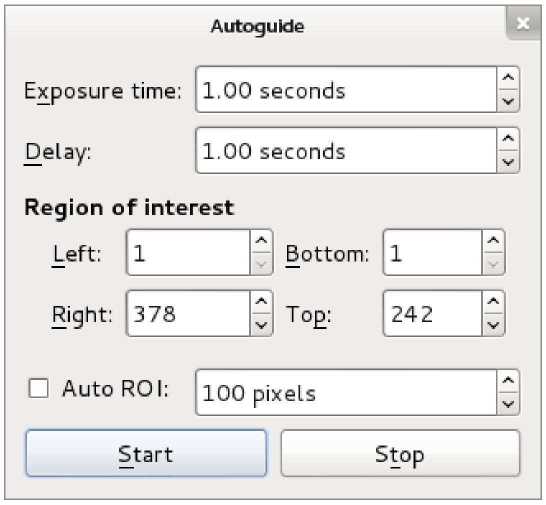


Figure 3.7: The autoguiding setup window.

Delay The amount of time to wait between guide images.

Left, Right, Top, Bottom The region of interest parameters for the CCD.

Auto ROI If checked, a region of interest is automatically determined as described above, using the specified box size for the subimage.

The "Start" button is used to begin the autoguiding process. Select the "Stop" button when autoguiding is no longer needed. If an exposure is in progress, it will be aborted.

The autoguiding plots window (Figure 3.8) displays several characteristics of the guide star measurements. The plot window displays time on the X-axis; ten minutes of guide history are shown. The X-axis is labeled with the UTC. The following quantities are plotted, from top to bottom:

- The guide offset in arcseconds, in the χ (with white points) and η (with yellow points) axis. The χ and η variables represent the coordinate axis of the tangent plane of the celestial sphere at the telescope's pointing location. For most purposes these axes are equivalent to right ascension and declination, respectively.
- The total stellar flux in digitized numbers.
- The full width of the stellar profile at half-maximum (FWHM). *The FWHM is displayed in non-physical units.*
- The ellipticity of the stellar profile.
- The phase angle of the stellar profile in degrees.

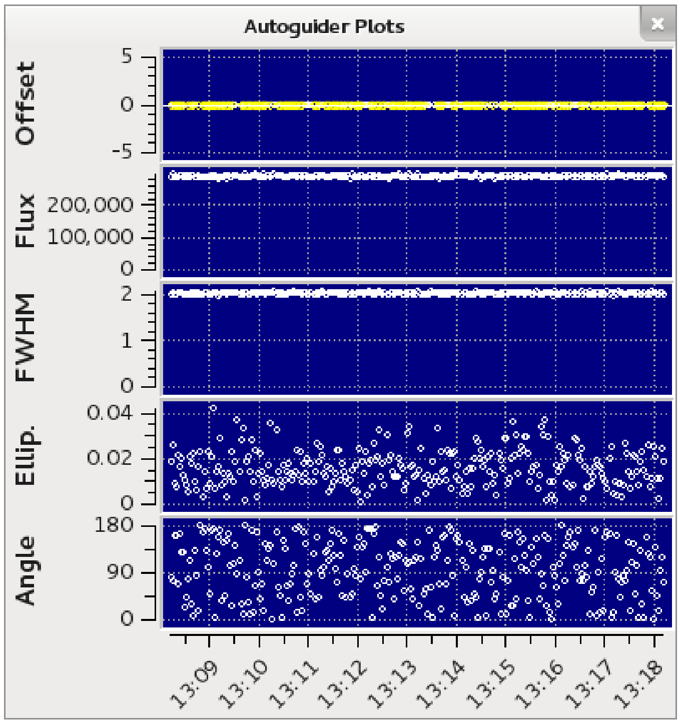


Figure 3.8: The autoguiding plots window.

3.4 Slewing a Telescope

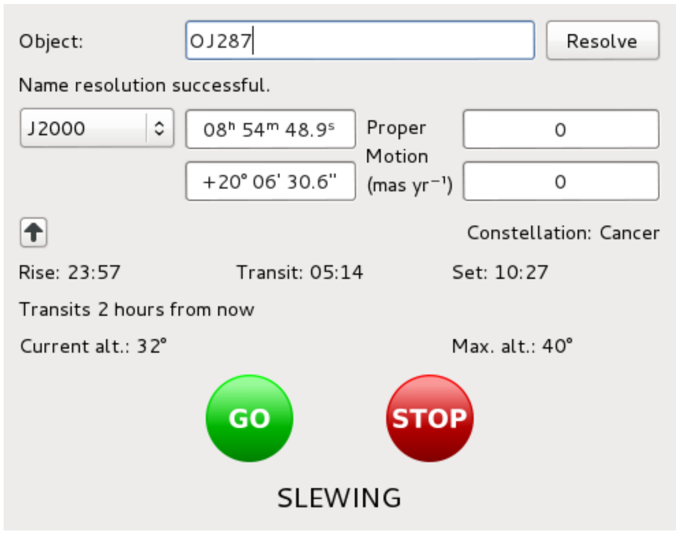
To move the telescope to a specified position, use the telescope slew tool.

The slew tool can automatically resolve object names to coordinates using the SIMBAD name resolver. To do this, type the identifier into the Object text box at the top of the tool, then press enter or select the "Resolve" button. If coordinates are available for the requested object, they are loaded in the fields below. If coordinates cannot be obtained, an error message is displayed below the text box.

Objects can be selected manually in the area below the object resolver area. Select the desired equinox using the combo box, then type the right ascension in the top left box, and the declination in the bottom left box. If you know the proper motion of the target, type the proper motion in right ascension into the top right box and the proper motion in declination into the bottom right box. Use units of milli-arcseconds per year.

Information about the selected target is displayed below the coordinate entry area. The information provided includes:

- The constellation where the object is located
- The time of day when the object last rose
- The time of day of the nearest transit of the object


Object: OJ287 Resolve
Name resolution successful.
J2000 | 08ʰ 54ᵐ 48.9ˢ | +20° 06′ 30.6″ | Proper Motion (mas yr⁻¹) 0 0
Constellation: Cancer
Rise: 23:57 Transit: 05:14 Set: 10:27
Transits 2 hours from now
Current alt.: 32° Max. alt.: 40°

SLEWING

Figure 3.9: The telescope slew tool

- The time of day when the object will next set
- The time since / until the rise / set / transit of the object, depending on which is nearest in time
- The current altitude of the object above the horizon
- The transit altitude of the object

To send the telescope to the target position, select the “Go” button. To stop the slew, select the “Stop” button.

The current status of the telescope is displayed at the bottom of this tool. This may be one of: “Idle”, if the telescope is not tracking a command position, “Slewing”, if the telescope is moving to meet pointing demands, or “Tracking”, if the telescope is maintaining commanded pointing demands.

3.5 Controlling ACE SmartDome

The dome control tool handles manual control of the dome through an interface to ACE SmartDome (Figure 3.10).

The dome control tool is divided into three parts: the top portion of the tool shows the current status of the doors, the middle shows the azimuth of the dome, and the bottom indicates the status of the auto-shutdown condition.

3.5.1 Door status

The status of the door is indicated by a large image of a dome at the top left of the tool and by a line of text at the top of the tool. The image will show the dome as open or closed, depending on the state of the

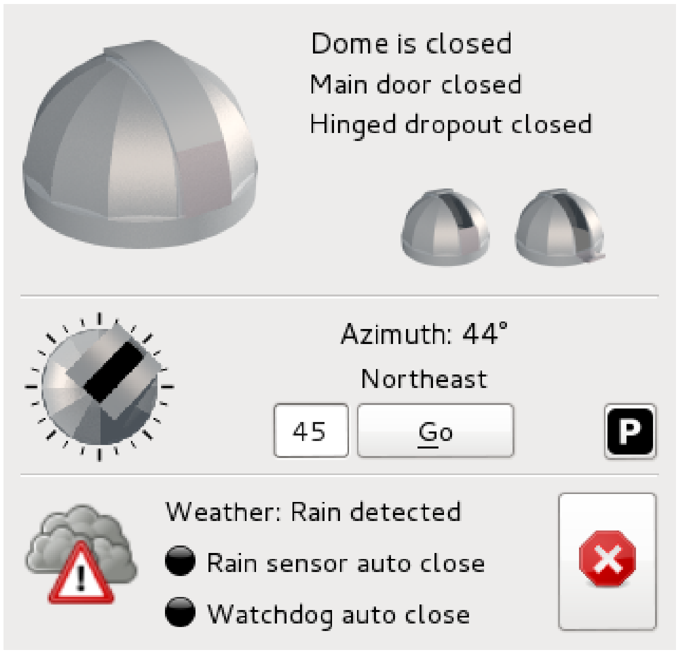


Figure 3.10: The dome control tool

dome. If the dome doors are moving, the image of the dome will pulsate. In domes that feature more than one door, the status of each door will be indicated below the first line of text. In the example, the status is shown as “Main door closed; Hinged dropout closed”.

Below the status text is a set of buttons that can be used to change the state of the dome doors. In the example, buttons are available to open the dome, either with the dropout raised or lowered.

3.5.2 Azimuth control

The dome azimuth is controlled in the second part of the tool. The azimuth is displayed as a position angle (degrees east of north) and graphically as a top-down perspective on the dome. The compass cardinal or ordinal direction of the dome is also displayed. To move to a new dome position, type the desired azimuth target into the text box and then press the enter key or select the “Go” button. The dome can also be moved to a pre-configured “park” position by selecting the button with the “P” icon.

Note: The azimuth reported for the dome may not be the same as the azimuth of the target object. If the optical axis of the telescope is not centered in the dome, an offset may be needed to keep the telescope in alignment with the dome. This is normal.

3.5.3 Auto-shutdown

SmartDome will automatically close under certain conditions to avoid damage to the telescope. Some of these can be viewed from within the interface.

The state of SmartDome's rain sensor is displayed in the dome control tool. If the rain sensor does not detect rain, an image of the moon is shown at the bottom left of the dome window, and the weather status message reads "OK to open". If rain is detected, clouds and a warning symbol appear and the message reads "Rain detected" (pictured).

The "Rain sensor auto close" indicator shows whether the dome will automatically close when the rain sensor detects rain. If this indicator is black, the telescope will not automatically respond to rain on the rain sensor. If the indicator is green, the dome will automatically close when rain is detected.

The "Watchdog auto close" indicator shows whether the dome will close automatically if communication between the SmartDome control electronics and the host computer is lost. If the host computer does not communicate with SmartDome for an extended period (by default five minutes) the dome will automatically close. If this feature is disabled, the indicator is shown as black. If enabled, the indicator is green.

The stop button is also shown in the bottom part of the dome tool. Pressing this button will stop all motion within the dome.

3.6 Filter Wheel Tool

The filter wheel tool (Figure 3.11) can be used to manually change the position of a filter wheel, or to modify the filter names or the focus offset settings.

A row of buttons is shown for each wheel. A green arrow at the left side of the window indicates the current position of the wheel. If the arrow is not visible, the wheel's current position is unknown. To move the filter to a new position, select the corresponding button. A spinner will appear next to that button, indicating that position as the wheel's destination. When the arrow points to the desired filter, the move is finished.

To change the filter names, select the tag icon in the window's lower right corner. Edit boxes will appear in place of the buttons (Figure 3.12). Change the names as desired, and finish editing by pressing the enter key or selecting the tag icon again.

To change the focus offset values, select the double-arrow icon in the window's lower right corner. Edit the focus offset values, and press enter or select the double-arrow icon again to finish.

3.7 Focuser Tool

This tool (Figure 3.13) allows manual control over the position of a focuser.

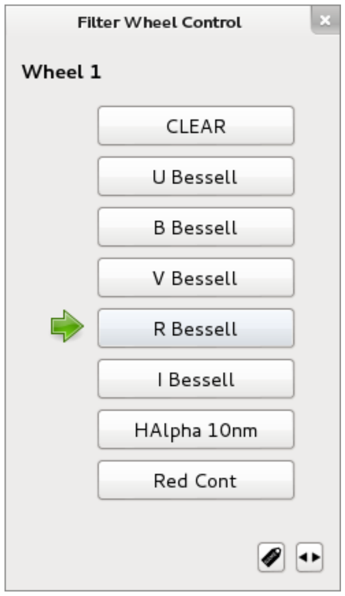


Figure 3.11: Filter wheel control tool

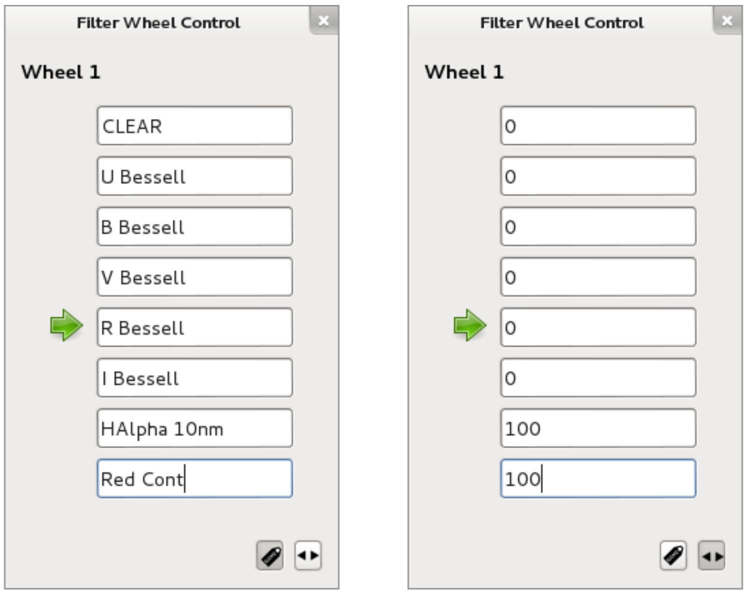


Figure 3.12: Filter wheel tool in editing modes

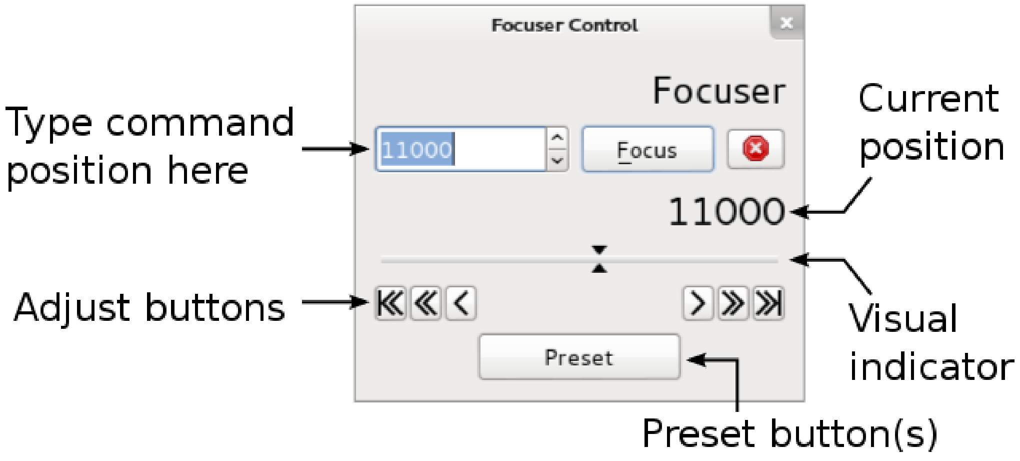


Figure 3.13: The manual focuser tool

To move the focuser to a specific position, type the desired position into the edit box and select the "Focus" button. The focuser may be stopped at any time by selecting the red "stop sign" button. The current position of the focuser is indicated by the large number below the focus and stop buttons. This number should begin changing when the Focus button is selected.

The visual indicator below the focus value illustrates the current and target position of the focuser with respect to the minimum and maximum positions. The target position is indicated by the arrow at the bottom, while the current position is indicated by the arrow at the top.

The six adjust buttons function as follows:

- 〉 Moves the focus forward by a small increment
- » Moves the focus forward by a large increment
- »| Moves the focus to the forward limit

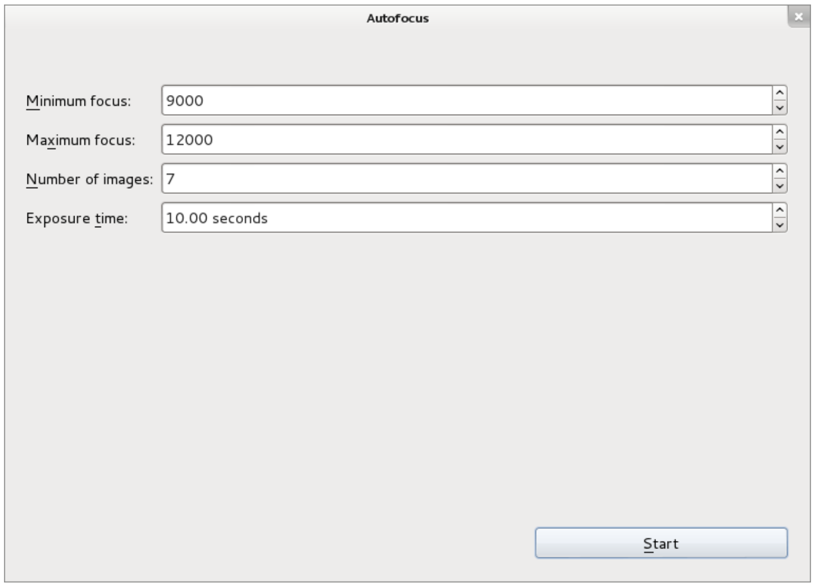
And similarly, the buttons with left arrows move the focuser in the reverse direction.

Note: Some of the adjust buttons may be unavailable in certain focuser models.

Preset focus values may be defined in the focuser's configuration. If presets are defined, they appear as buttons at the bottom of the focus control window. To move to a preset position, select that preset's button

3.8 AUTOFOCUS

The autofocus tool can be used to take a series of images at a range of focus values, and then analyze the images to determine the point of best focus.



Minimum focus: 9000
Maximum focus: 12000
Number of images: 7
Exposure time: 10.00 seconds

Figure 3.14: Entering the autofocus parameters

The autofocus tool works as a wizard with three steps. In the first step, shown in Figure 3.14, you enter information about how you want to perform the autofocus sequence:

Minimum focus The lowest focus value to try

Maximum focus The highest focus value to try

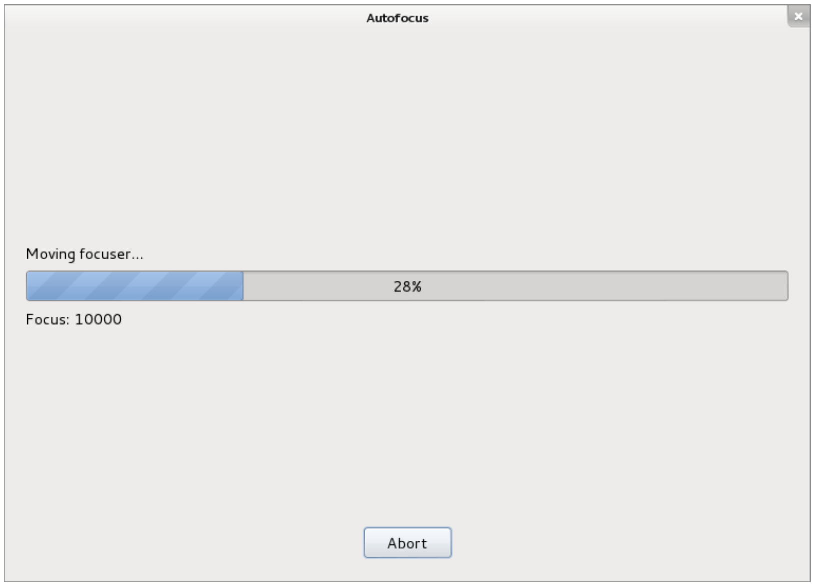
Number of images The number of images to take for the focus sequence. The images will be evenly spaced between the minimum and maximum focus value.

To begin the sequence, select the "Start" button. A progress bar will appear (Figure 3.15), indicating how far the sequence has progressed. The system will proceed to take a series of exposures at various focus values. When the sequence is finished, the resulting images will be analyzed to determine the best focus. If you want to stop the sequence, select the "Abort" button. You will be returned to the initial screen.

Once the autofocus sequence has completed (Figure 3.16), you will be presented with a graph of image FWHM *vs* Focus value. If the process was successful, you should see a parabolic curve in the focus data, and the minimum of the parabola will indicate the position of best focus.

The autofocus tool will attempt to identify the minimum and will move the focuser to its best guess. Examine the resulting data and verify that the focuser has moved to the correct position. If you wish to correct the best focus position, type a new best focus value into the entry box and select "Accept". To go back to the beginning without adjusting the focus value, select "Cancel".

For best results, use the autofocus tool with a range centered nearly as possible on the expected best focus position. Use a focus range large enough for a noticeable difference in focus to appear (Figure 3.15, but small enough that stars are clearly visible in all of the images. The best focus determined by the autofocus routine is likely to be very close, but often the measurement can benefit by manual examination of the

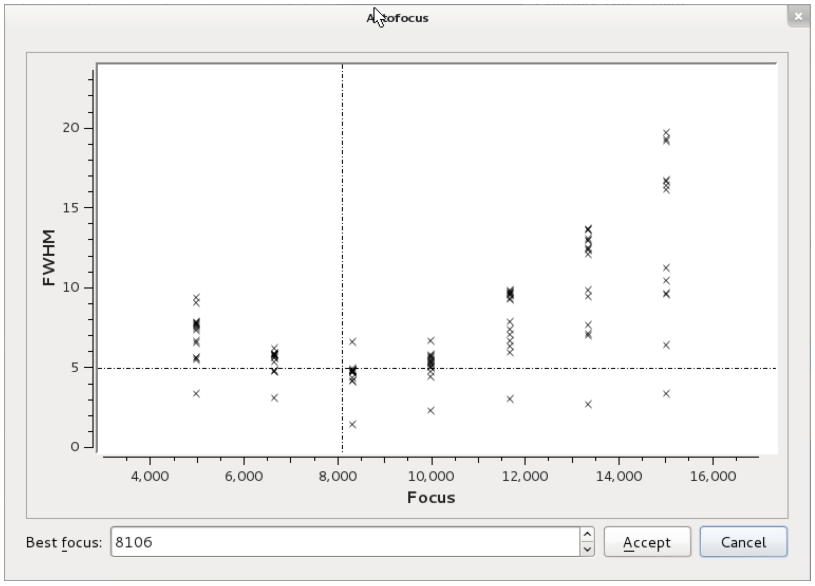


Moving focuser...

28%

Focus: 10000

Figure 3.15: The autofocus routine collecting its data



Best focus: 8106

Figure 3.16: The focus results showing a best focus of 8106

resulting graph.

3.9 Switch Panel

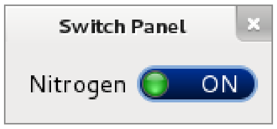


Figure 3.17: A switch controlling a nitrogen valve

A switch device can be used to turn an instrument on or off. All of the switch devices associated with a site appear in the Switch Panel (Figure 3.17). The switch panel features an on/off switch labeled "ON" or "OFF" for each of the switch devices that can be controlled. To change the state of a device, select the switch for that device. The label should change from "OFF" to "ON" or *vice versa*.

4 C++ Interface

4.1 Introduction

You can use Connector's C++ SDK to create your own programs that interface with and control the instruments made available to Connector. This chapter describes how to build these programs.

To create a program that interfaces with Connector, you will need to link to the Connector common library. You will also need to reference Connector's header files. These should be included with your installation. You may also need to link to CFITSIO and certain Boost libraries.

Your program needs to contain at least one instance of the `ace::MainLooper` class. Each instance represents one thread in which commands may be executed. Your program should generally have a line like this:

```
ace::MainLooper looper[4]
```

This would create four threads for command execution. Your instances of the `MainLooper` class must remain in scope until your program is finished using Connector.

Instrumentation controlled by Connector is accessed through interfaces. To use an instrument, you will need to obtain a handle. To do this:

1. Establish a connection to the server you wish to use.
2. Obtain a handle for the node that hosts the instrument you need.
3. Use the `ace::Peer::getInstrument` method, using the desired interface as the template argument, for example:

   ```
   CameraPtr camera = peer->getInstrument<ace::Camera>("My Camera")
   ```

   to use an instrument named "My Camera" with the `Camera` interface.

Connector makes heavy use of the Boost Signals2 library to provide signals. You can make use of these signals to receive notifications when a significant event occurs in the connector system. For more information about the Signals2 library, go to http://www.boost.org/doc/libs/1_57_0/doc/html/signals2.html.

4.2 Core functionality

typedef boost::function< void(int32_t)> **`CallbackFunction`**

ResultCode enum

Codes for supplying to a callback function.

Values:

- `CMD_SUCCESS` - The command completed successfully.
- `CMD_INVALID` - The command was invalid as submitted.
- `CMD_ABORTED` - The command was aborted by the user, possibly through an emergency stop.
- `CMD_DISABLED` - The command failed because a required instrument has not been enabled.
- `CMD_CONNECT_ERROR` - The command failed because a network or other communications link was lost.
- `CMD_EQUIPMENT_ERROR` - The command failed because of an apparent hardware error.
- `CMD_OTHER_ERROR` - The command failed because of a software or other electronics issue (e.g., full hard disk)
- `CMD_LIMIT` - The command failed because motion was interrupted by a limit switch.
- `CMD_RESTRICTION` - The command failed because of a hardware restriction other than a limit switch (e.g., sun avoidance)
- `CMD_UNKNOWN_CONDITION = 15` - None of the above applies.

class **`ace::Node`**

Public Functions

PeerPtr **`getPeer`**(uuid *id*)
Returns the peer specified by the given UUID.

PeerPtr **`getPeer`**(std::string *name*)
Returns the peer specified by the given name.

boost::uuids::uuid **`id`**()
Returns the identifier ID for this node.

std::string **`info`**() const
Returns a human-readable information string about the node status.

class **`ace::Peer`**

4.3 INSTRUMENT INTERFACES

typedef boost::shared_ptr< *Telescope* > **`TelescopePtr`**

class **`ace::Telescope`**
Handles the motion of a telescope required for the acquisition of a target.

Public Functions

virtual void **`goTo`**`(`const CelestialPositionPtr *pos*, CallbackFunction *onCompletion*`)` = 0
Sends the telescope to the specified position.

virtual void **`stop`**`()` = 0
Stops any slewing or other motion in the telescope, and disables tracking.

virtual CelestialPositionPtr **`getPosition`**`()` const = 0
Returns the current position of the telescope.

virtual std::string **`getRaJ2000`**`()` const = 0
Returns the current RA for epoch J2000.

virtual std::string **`getDecJ2000`**`()` const = 0
Returns the current declination for epoch J2000.

virtual CelestialPositionPtr **`getTarget`**`()` const = 0
Returns the position to which the telescope is currently moving.

virtual void **`nudge`**`(`Direction *dir*`)` = 0
Nudges the telescope in the specified direction.

Motion continues until *stopNudge()* is called, or until five seconds have elapsed, or when a limit is encountered, whichever comes first. Unlike most interface calls, this call is not idempotent, in that calling *nudge()* again will reset the timer.

Parameters

- `dir` - the direction to nudge the telescope.

virtual void **`stopNudge`**`()` = 0
Cancels a move caused by *nudge()*.

If a call to *nudge()* is not currently in effect, this method does nothing.

virtual double **`nudgeSpeed`**`()` const = 0
Sets the speed for telescope motion when the *nudge()* command is called.

virtual void **`nudgeSpeed`**`(`double *sp*`)` = 0
Returns the speed used for *nudge()* commands.

virtual double **`minNudgeSpeed`**`()` const = 0
The minimum accepted speed for “nudging” the telescope.

Calls to *nudgeSpeed(double)* must use a parameter greater than the returned value.

Return

the minimum nudge speed.

virtual double **maxNudgeSpeed**() const = 0
The maximum accepted speed for nudging the telescope.

virtual void **guide**(double *deltaHa*, double *deltaDec*, CallbackFunction *onCompletion*) = 0
Moves the telescope by the specified amount, to guide the telescope.

Parameters

- `deltaHa` - Hour angle offset to apply to the telescope position, in degrees.
- `deltaDec` - Declination offset to apply to the telescope position, in degrees.
- `onCompletion` - a callback function called when the guide offset is completed.

virtual WorldCoordinates **getWcs**() const = 0
Returns a structure that can be used to compute WCS headers for a FITS image.

Public Members

boost::signals2::signal< void(double, double)> **pointingChanged**
Emitted when the telescope's position (RA and Dec) have changed.

typedef boost::shared_ptr< *Camera* > **CameraPtr**

class **ace::Camera**
An interface for basic CCD cameras.

Public Type

State enum
Possible states of the camera.

Values:

- `IDLE` - The camera is powered on and ready to expose.
- `EXPOSING` - The camera is exposing.
- `PAUSED` - An exposure is paused and can be resumed.

Public Functions

virtual *State* **getState**() const = 0
Returns the current camera status.

virtual void **`getCameraDimensions`** (int & *x*, int & *y*) const = 0
: Calling this function will fill the x and y parameters with the number of pixels on the CCD.

virtual void **`getPixelSize`** (double & *x*, double & *y*) const = 0
: Calling this function fills the x and y parameters with the size of the pixels in the CCD, in microns.

virtual double **`getCcdSetpoint`** () const = 0
: Returns the set point for the CCD cooler temperature.

virtual void **`setCcdSetpoint`** (double *t*) = 0
: Sets the CCD temperature set point to t.

virtual double **`getCcdTemperature`** () = 0
: Returns the current temperature of the CCD.

virtual void **`exposeSequence`** (const Exposure & *exp*, uint32_t *nExposures*, CallbackFunction *onCompletion*) = 0
: Starts a sequence of exposures based on a template.

virtual void **`expose`** (const Exposure & *exp*, CallbackFunction *onCompletion*) = 0
: Starts an exposure.

virtual bool **`canPause`** () const = 0
: Returns true if an exposure can be paused and resumed with *pause()* and *resume()*.

virtual bool **`pause`** () = 0
: Pauses the exposure without readout.

This method may throw if fewer than ten seconds are left in the exposure.

Return

true if the camera successfully paused, or false otherwise

virtual void **`resume`** () = 0
: Resumes a paused exposure.

virtual void **`abort`** () = 0
: Aborts the currently-in-progress exposure.

virtual Exposure **`getExposure`** () const = 0
: Returns the currently in-progress exposure, if one is being taken, or the most recently taken exposure if the camera is idle.

If no exposure has been taken since system start, an exception is thrown.

virtual int **`bitsPerPixel`** () const = 0
: Returns the number of bits per pixel in the resulting image.

Usually this is 16.

virtual void **`getPixels`** (int *x1*, int *y1*, int *x2*, int *y2*, int *xSkip*, int *ySkip*, char * *pixels*) const = 0

Returns the image pixels.

Parameters

- `x1` - the left side of the subimage to retreive
- `y1` - the bottom side of the subimage
- `x2` - the right side of the subimage
- `y2` - the top of the subimage
- `xSkip` - only return every xSkip column of the subimage
- `ySkip` - only return every ySkip row of the subimage
- `pixels` - area where the image pixels should be copied

virtual void **`listReadoutModes`** (std::vector< std::string > & *types*) const = 0

Lists the available readout modes.

The readout mode setting affects how the image pixels are read from the CCD to the amplifier. For example, many CCDs have four amplifiers and can read out from any or all of them.

Parameters

- `types` - A list of names of the different readout modes available.

virtual void **`setReadoutMode`** (std::string *mode*) = 0

Sets the current CCD readout mode.

Parameters

- `mode` - The name of the readout mode to use in future exposures.

virtual std::string **`getReadoutMode`** () const = 0

Returns the current CCD readout mode.

virtual HeaderCardList **`getHeaders`** () const = 0

Returns the list of items used to construct the FITS header.

virtual void **`setHeaders`** (const HeaderCardList & *list*) = 0

Sets the list of items used to construct the FITS header.

Parameters

- `list` - the new header card list

virtual void **`getFitsFile`** (void ** *data*, uint32_t & *nbytes*) const = 0

Returns the contents of the most recently created FITS file.

Parameters

- `data` - a pointer to the contents of the fits file. The contents of this pointer should be released by the caller by a call to free() after the data are no longer needed.
- `nbytes` - the number of bytes contained in the data pointer

virtual OpticalAxis **`getOpticalAxis`**`()` const = 0
Returns information about the optical axis the camera is attached to.

virtual void **`setOpticalAxis`**`(`const OpticalAxis & *oa*`)` = 0
Sets the optical axis information.

virtual uint32_t **`getRemaining`**`()` const = 0
Returns the number of exposures to be taken after the current exposure is complete.

Public Members

boost::signals2::signal< void(*State*)> **`stateChanged`**
Fired when the camera status changes.

boost::signals2::signal< void()> **`exposureStarted`**
Fired when an exposure has been started.

boost::signals2::signal< void()> **`exposureComplete`**
Fired when an exposure and readout have successfully completed.

boost::signals2::signal< void()> **`exposureAborted`**
Fired when an exposure was aborted by the user before completion.

boost::signals2::signal< void()> **`exposureFailed`**
Fired when an exposure has failed because of a problem in the camera or the control software.

boost::signals2::signal< void(double, double, bool)> **`progressChanged`**
Fired when new data are available about the completion progress of an exposure.

boost::signals2::signal< void()> **`fitsReady`**
Fired when an image can be obtained with getFitsFile.

boost::signals2::signal< void(uint32_t)> **`remainingChanged`**
Fired after a change in the number of exposures queued as a result of a call to *exposeSequence()*.

typedef boost::shared_ptr< *ImageWriter* > **`ImageWriterPtr`**

class **`ace::ImageWriter`**
An interface for writing CCD images to disk.

Public Functions

virtual void **`setRoot`** (const std::string & *rootPath*) = 0

Sets the root directory that images will be written into.

The writer must not write anywhere except this directory and its subdirectories. If an empty string is used for the root path, images can be written anywhere, on any disk on the system.

Parameters

- `rootPath` - the absolute path for new images.

virtual void **`setProcessPath`** (const std::string & *processPath*) = 0

Sets the directory for processing of images.

virtual std::string **`getRoot`** () const = 0

Returns the root directory that images are written into.

virtual std::string **`getProcessPath`** () const = 0

Returns the root directory in which images are processed.

virtual void **`setProcessScript`** (uint8_t *type*, const std::string & *command*) = 0

Sets the name of the script that will be used to process images.

See

Exposure::Type

Parameters

- `type` - the image type
- `command` - the command to be executed. The command will be given two arguments: the path to the processing root, and the name of the newly created image.

virtual std::string **`getProcessScript`** (uint8_t *type*) const = 0

Returns the name of the processing script for the given image type.

See

Exposure::Type

virtual void **`setFilenameTemplate`** (const std::string & *t*) = 0

Sets the template for filenames for images.

The template may contain patterns like "{seq}" for the current sequence number.

Parameters

- `t` - the new template string

virtual std::string **`getFilenameTemplate`**`()` const = 0
Returns the current filename template for new images.

virtual void **`setNextSequenceNumber`**`(`int *n*`)` = 0
Sets the sequence number for the next exposure.

virtual int **`getNextSequenceNumber`**`()` const = 0
Returns the sequence number that will be applied to the next exposure.

virtual void **`saveFile`**`(`const std::string & *filename*`)` = 0
Saves a copy of last image immediately.

Parameters

- `filename` - The filename for the image. The image will be saved in the subdirectory specified to *setRoot()*.

virtual void **`getFitsFile`**`(`void ** *data*, uint32_t & *nbytes*`)` const = 0
Returns the contents of the most recently created FITS file.

Parameters

- `data` - a pointer to the contents of the fits file. The contents of this pointer should be released by the caller by a call to free() after the data are no longer needed.
- `nbytes` - the number of bytes contained in the data pointer

Public Members

boost::signals2::signal< void(const std::string &)> **`templateChanged`**
Emitted when the filename template has changed through a call to setFilenameTemplate.

boost::signals2::signal< void(const std::string &)> **`rootPathChanged`**
Emitted when the root path has changed through a call to setRootPath.

boost::signals2::signal< void(const std::string &)> **`imageSaved`**
Emitted when a new image has been saved to the disk.

boost::signals2::signal< void(const std::string &)> **`imageSaveError`**
Emitted when an image could not be saved because an error occurred.

boost::signals2::signal< void()> **`fitsFileReady`**
Emitted when a new FITS file is available.

typedef boost::shared_ptr< *FilterWheel* > **`FilterWheelPtr`**

class **`ace::FilterWheel`**
Represents a filter wheel.

A filter wheel can be used to select any of a group of filters, and introduce a filter into the light path. The filters must be numbered in some way. This class expects the filter wheel to number the filters from zero to n-1.

Public Type

State enum

States for filter wheel activity.

See

FilterWheel::getState()

Values:

- `UNKNOWN` - The wheel's state is unknown, or none of the.
- `STOPPED` - other states apply

  The wheel is stopped at a valid position
- `INITIALIZING` - The wheel is initializing.
- `MOVING` - The wheel is moving to a new position.
- `FAULT` - The wheel is not in a valid state.

Public Functions

virtual *State* **`getState`**`()` const = 0

Returns the current state of the filter wheel.

See

FilterWheel::State

virtual void **`init`**`(`CallbackFunction *func*`)` = 0

Initializes the wheel.

virtual void **`stop`**`()` = 0

Stops all motion in the wheel.

virtual void **`goTo`**`(`int32_t *pos*, CallbackFunction *onCompletion*`)` = 0

Sends the wheel to the specified position, where 0 is the first filter position.

virtual void **`goTo`**`(`const std::string & *name*, CallbackFunction *onCompletion*`)` = 0

Sends the wheel to the position identified by name.

Parameters

- `name` - the name of the target filter for the wheel
- `onCompletion` - function to call when the move is finished

virtual int32_t **`getPosition`**`()` const = 0
Returns the current position of the filter wheel.

virtual std::string **`getFilter`**`()` const = 0
Returns the name of the filter currently in position.

virtual int32_t **`getTarget`**`()` const = 0
Returns the current target of the wheel.

If the current move completes successfully, this will be the wheel's position.

virtual uint32_t **`countPositions`**`()` const = 0
Returns the number of positions available in this wheel.

virtual void **`setFocusOffsets`**`(`const std::vector< double > & *offsets*`)` = 0
Sets the focus offsets for each filter.

When offsets are nonzero, moving the filter wheel should also cause the focuser to move relative to its current position so that the image remains focused appropriately for the current filter.

Pre

offsets.size() == *countPositions()*

Parameters

- `offsets` - the offset values to use for future filter changes

virtual void **`getFocusOffsets`**`(`std::vector< double > & *offsets*`)` const = 0
Returns the focus offset values.

When the filter wheel position is changed from i to j, the focus value should change by an amount determined by offsets[i] - offsets[j].

Post

offsets.size() == *countPositions()*

Parameters

- `offsets` - a vector where the offset values will be stored.

virtual double **`getFocusOffset`**`(`int32_t *pos*`)` const = 0
Returns the focus offset for a specific wheel position.

Return

the focus offset value for position pos

Parameters

- `pos` - the position number for which to lookup the offset

virtual void **`setNames`** (const std::vector< std::string > & *names*) = 0
Sets the names to use for the filters.

These names can be used in calls to *goTo(const std::string&,CallbackFunction)*

Pre

names.size() == *countPositions()*

Parameters

- `names` - names to use for the filters in this wheel

virtual void **`getNames`** (std::vector< std::string > & *names*) const = 0
Returns the names of the filters in this wheel.

The names will be stored into the vector provided by *names*

Post

names.size() == *countPositions()*

Parameters

- `names` - a vector where the filter names will be stored.

Public Members

boost::signals2::signal< void(*State*)> **`stateChanged`**
Fires when the filter wheel's state (as reported by *getState()*) has changed.

boost::signals2::signal< void(int32_t)> **`positionChanged`**
Fires when the filter wheel has moved to a new position.

boost::signals2::signal< void(int32_t)> **`targetChanged`**
Fires when the filter wheel's target position has changed.

When a move command finishes (either successfully or unsuccessfully) the signal is fired again with an argument of -1.

boost::signals2::signal< void()> **`namesChanged`**
Fires when the filter names have been changed.

boost::signals2::signal< void()> **`offsetsChanged`**
Fires when the focus offset for a filter position has changed.

typedef boost::shared_ptr< *Focuser* > **`FocuserPtr`**

class **`ace::Focuser`**

An interface for controlling a focuser.

The focuser can be moved continuously between two focus extremes given by *Focuser::getMinimum()* and *Focuser::getMaximum()*. Focusers also have a preset list of positions. One can easily recall a frequently used position by calling *goToPreset()* using a previously defined preset position name.

Public Type

State enum

Possible states for the focuser.

Values:

- `STOPPED` - The focuser is stopped and in position.
- `LIMIT_REV` - The focuser is at its reverse limit.
- `LIMIT_FWD` - The focuser is at its forward limit.
- `MOVING_REV` - The focuser is moving in the reverse direction.
- `MOVING_FWD` - The focuser is moving in the forward direction.
- `MOVING` - The focuser is moving, but the direction is.

  unspecified.

Public Functions

virtual *State* **`getState`**`()` const = 0

Returns the current focuser state.

virtual void **`init`**`(`CallbackFunction *onCompletion*`)` = 0

Begins the focuser initialization process.

virtual void **`go`**`(`double *position*, CallbackFunction *onCompletion*`)` = 0

Sends the focuser a command to move to the specified position.

Parameters

- `position` - the position to which to move

virtual void **`stop`**`()` = 0

Immediately stops the focuser.

virtual void **`definePosition`** (double *position*) = 0
Sets the current position of the focuser to the value specified.

The focuser is not physically moved.

Parameters

- `position` - the new value for the focus position

virtual double **`getFocusPosition`** () const = 0
Reports the current position of the focuser.

virtual double **`getCommandedPosition`** () const = 0
Reports the last position that the focuser has been commanded to move to by a call to *go()*.

virtual double **`getMinimum`** () const = 0
Returns the minimum focus value.

The focuser must be able to go to this position without hitting the reverse limit.

virtual double **`getMaximum`** () const = 0
Returns the maximum focus value.

The focuser must be able to go to this position without hitting the forward limit.

virtual void **`goToMinimum`** (CallbackFunction *onCompletion*) = 0
Sends the focuser to the reverse limit.

This function ignores the limitations specified by *getMinimum()*.

virtual void **`goToMaximum`** (CallbackFunction *onCompletion*) = 0
Sends the focuser to the forward limit.

This function ignores the limitations specified by *getMaximum()*.

virtual void **`setPreset`** (const char * *name*, double *position*) = 0
Creates a new preset position of the specified focus value.

If the name given already exists as a preset value, it is updated to the specified value.

Once a preset value is created, the focuser can be moved to a preset position by a call to goToPreset(name).

Parameters

- `name` - the name of the preset position to create or update
- `position` - the focus position for this preset

virtual void **`removePreset`** (const char * *name*) = 0
Removes the specified item from the list of preset positions.

Parameters

- `name` - the name of the item to remove

virtual void **`goToPreset`** (const char * *name*, CallbackFunction *onCompletion*) = 0
Sends the focuser to the preset position specified by name.

Parameters

- `name` - the name of the preset position to move to.

virtual void **`listPresets`** (std::vector< std::string > & *presets*) const = 0
Returns a set containing all of the names for preset values available for the focuser.

Parameters

- `presets` - preset names will be added to this set

virtual double **`getPreset`** (const char * *name*) const = 0
Returns the focus position corresponding the preset value specified by name.

A call to goToPreset(name) is equivalent to go(getPreset(name)).

Parameters

- `name` - the value of the preset position to retrieve.

virtual void **`setPresets`** (const std::list< std::pair< std::string, double > > & *presets*) = 0
Clears the list of presets and replaces it with the given list of preset focus positions.

Parameters

- `presets` - the new list of presets for the focuser

Public Members

boost::signals2::signal< void(*State*)> **`stateChanged`**
The state of the focuser changed.

boost::signals2::signal< void()> **`initializeStarted`**
A command to initialize the focuser was received.

boost::signals2::signal< void()> **`initialized`**
The focuser has been initialized successfully.

boost::signals2::signal< void()> **`aborted`**
A command to move the focuser was aborted by the user.

boost::signals2::signal< void()> **failed**
: A command to move the focuser failed because of a mechanical or communications problem.

boost::signals2::signal< void(double)> **moveStarted**
: The focuser has been commanded to move by a call to go(double)

boost::signals2::signal< void(double)> **moveCompleted**
: The focuser has finished a move commanded by a call to go(double)

boost::signals2::signal< void(double)> **commandedPositionChanged**
: The position that the focuser should move to has changed.

boost::signals2::signal< void(double)> **positionChanged**
: The position reported by the focuser has changed.

boost::signals2::signal< void()> **limitReached**
: A limit has been reached.

4.4 Example C++ Client Program

This program demonstrates use of the C++ interface to connector. The program points a telescope to each of three targets. For each target, it takes a series of exposures in the UBVRI filter set.

This program illustrates use of the `Event` class. This class provides a `wait()` method which blocks until the callback is called. You can create your own objects to receive callbacks for more sophisticated responses to command completion events. Use of signals is also not demonstrated in this example.

```
#include <iostream>

#include <ace/core/Node.h>
#include <ace/core/MainLooper.h>
#include <ace/core/RaDecPosition.h>
#include <ace/core/Event.h>

#include <ace/interfaces/Camera.h>
#include <ace/interfaces/ImageWriter.h>
#include <ace/interfaces/Telescope.h>
#include <ace/interfaces/FilterWheel.h>

struct target_t
{
    const char* name;
    double ra;
    double dec;
};
```

```
int main()
{
    const char* filters[] = { "U", "B", "V", "R", "I" };
    double exptimes[] =     { 30,  20,  15,  10,  5   };
    const int nExposures = 3;

    target_t targets[] = {
        { "M37",            88.075,        32.5533 },
        { "NGC3521",        166.4524708,   -0.0359 },
        { "Eskimo Nebula", 112.29485793, 20.911800825 }
    };

    // You must construct at least one instance of
    // ace::MainLooper. This class handles the thread
    // pool used to process activity in the client.
    ace::MainLooper loop[8];

    ace::Node n;
    ace::PeerPtr peer;

    ace::Event block;

    ace::CameraPtr camera;  // Represents a generic camera
    ace::ImageWriterPtr writer;  // Represents an image archiver
    ace::TelescopePtr telescope; // Represents an astronomical telescope
    ace::FilterWheelPtr wheel;   // Represents a filter wheel

    n.connect("localhost", 9889);
    peer = n.getPeer("Instrument Server");

    // Peer::getInstrument attaches the instrument objects to
    // specific instruments contained in your system. These calls
    // do a lookup to determine the instruments' handle IDs,
    // and associates the objects with the IDs for those instruments.
    camera = peer->getInstrument<ace::Camera>("My Camera");
    writer = peer->getInstrument<ace::ImageWriter>("My Camera");
    telescope = peer->getInstrument<ace::Telescope>("My Telescope");
    wheel = peer->getInstrument<ace::FilterWheel>("My Filter Wheel");


    /* The ImageWriter interface is built into most cameras. This
     * interface can be used to automatically write images after they
     * are taken with the camera.
     */
```

```
/* The filename template determines how image files will be named
 * when they are saved into the archive. These files will be stored
 * on the camera computer in a previously specified directory.
 */
writer->setFilenameTemplate("{{object}} - {{filter}} - {{seq}}.fits");

for (int i = 0; i < sizeof(targets) / sizeof(target_t); ++i)
{
    /* The Camera::Exposure structure is used to set parameters
     * for an exposure. This structure is then passed into the
     * expose() or exposeSequence() method.
     */
    ace::Exposure exp;
    exp.object = targets[i].name;   // used for writing to the header

    /* Sends the telescope to the specified coordinates. */
    std::cout << "Moving the telescope" << std::endl;
    ace::CelestialPositionPtr pos(
        new ace::RaDecPosition(targets[i].ra, targets[i].dec));
    telescope->goTo(pos, block);

    /* The ace::Event class is a convenient way to cause
     * execution to block until a method calls its callback.
     * Here, we block further execution until the
     * Telescope::goTo method is complete.
     */
    ace::ResultCode result = block.wait();

    std::cout << "Result of Telescope::goTo operation was: " <<
        result << std::endl;

    for (int j = 0; j < sizeof(filters)/sizeof(char*); ++j)
    {
        exp.exptime = exptimes[j];

        std::cout << "Sending wheel to " << filters[j] << std::endl;
        wheel->goTo(filters[j], block);
        block.wait();

        std::cout << "Beginning an exposure sequence" << std::endl;
        // You can use Camera::exposeSequence when you
        // have many identical exposures.
        camera->exposeSequence(exp, nExposures, block);
        block.wait();
    }
    }

    std::cout << "Finished." << std::endl;
        return 0;
}
```

5 Python Scripting Interface

5.1 Establishing a Connection with Python

class `ace.syscore.`**`AceConnection`**

Connections between a Python instance and ACE Connector nodes are managed by the Python class `ace.syscore.AceConnection`. When using an instrument with Ace Connector, the first step is always to construct an `AceConnection` object to manage the connection.

5.1.1 Construction

`ace.syscore.`**`AceConnection`**(*host*, *port=9889*)

Establishes a connection to a Connector node identified by the DNS hostname or IP address specified by *host* listening on the port specified by *port*.

Parameters

- **host** (*string*) – hostname to connect to
- **port** (*integer*) – port number

5.1.2 Obtaining Instrument Control from a Connection

Once a connection is established, you will need to access an instrument interface. To control an individual instrument accessible from a connection, you need to know:

1. The interface that provides the desired functionality
2. The name of the node that hosts the desired instrument
3. The name that identifies the instrument to the system

For example, your connector setup might contain a node named `Telescope PC`, hosted on a computer with DNS name `tel.example.net` on the default port and an instrument named `Guider Focuser`. You might want to use the instrument with the `ace.focuser.Focuser` interface. This would allow you to determine the current focus position and move to new focus positions. To access this interface, you would use the following:

```
>> import ace.syscore, ace.focuser
>> conn = ace.syscore.AceConnection('tel.example.net')
>> focuser = ace.focuser.Focuser(conn, 'Telescope PC', 'Guider Focuser')
```

And you may then use the functions associated with the `Focuser` class.

Similarly, to use the filter wheel interface:

```
>> import ace.filterwheel
>> wheel = ace.filterwheel.FilterWheel(conn, 'Telescope PC', 'Filter Wheel')
```

For more detail, see the *Example Python Script* at the end of this chapter.

5.2 Camera Interface

5.2.1 Camera State Constants

class `ace.camera.`**`state`**

`IDLE`
No exposure is currently in progress.

`EXPOSING`
An exposure is currently under way. This status is also reported while the camera is reading out.

`PAUSED`
The camera is paused. The exposure can be resumed by a call to `Camera.resume()`, or cancelled by a call to `Camera.abort()`.

5.2.2 Exposure Type Constants

class `ace.camera.`**`exposure_type`**

`LIGHT`
An exposure with the shutter open. This is also known as an "object" or "science" exposure.

`FLAT`
An exposure in which the telescope is pointed at a flat field screen, for calibration.

`DARK`
An exposure in which the shutter remains closed during the exposure, so that dark current can be measured. Dark frames generally have a nonzero exposure time.

`BIAS`
An exposure with zero exposure time, for measurement of the bias level.

`SKYFLAT`
A flat field exposure collected by pointing the telescope into the twilight sky.

5.2.3 Control Functions and Data

class `ace.camera.`**`Camera`**

`state`
Current camera state, either `IDLE`, `EXPOSING`, or `PAUSED`.

`temperature`
Current temperature of the CCD.

`setpoint`
Setpoint (target) temperature for the CCD.

`readout_mode`
The currently selected readout mode. This setting can be used to control readout rate, amplifier selection, etc., according to what is supported by the camera.

`expose`(*exptime*, *type=LIGHT*, *bin_x=1*, *bin_y=1*, *x1=1*, *x2=1*, *y1=1*, *y2=1*, *overscan=0*, *save=True*, *block=True*)
Starts a new exposure.

Parameters

- **exptime** – exposure time in seconds
- **type** – Type of exposure to take
- **bin_x** – pixel binning in the x-axis
- **bin_y** – pixel binning in the y-axis
- **x1** – Left edge of the imaging subregion, in unbinned pixels
- **x2** – Right edge of the imaging subregion, in unbinned pixels
- **y1** – Bottom edge of the imaging subregion, in unbinned pixels
- **y2** – Top edge of the imaging subregion, in unbinned pixels
- **overscan** – Number of overscan columns
- **save** – Saves the image when complete
- **block** – Causes the function to block until the exposure is complete

`abort`()
Aborts the current exposure.

`pause`()
Causes the current exposure to pause. The shutter is closed and the camera's clock is paused. When `resume()` is called, the shutter is re-opened and the clock resumes.

Returns True if the command was successful, or `False` if the exposure could not be paused because readout is in progress or about to begin.

Return type boolean

`resume``()`
Resumes a paused exposure. If `pause()` has not been called, this function has no effect.

`can_pause`
`True` if the `pause()` function can be used. If `False`, the effect of calling `pause()` is undefined.

5.3 Telescope Interface

class `ace.telescope.`**`RaDecPosition`**
Represents a celestial position defined by a constant right ascension and declination, to describe objects that move sidereally.

`ra`
Right ascension specified in degrees. This should be between 0 and 360.

`dec`
Declination specified in degrees. This should be between -90 and +90.

class `ace.telescope.`**`Telescope`**

`go_to``(`*pos*`)`
Sends the telescope to the specified position.

Parameters **pos** (*CelestialPosition*) – The new target position.

`go_to_j2000``(`*ra*, *dec*`)`
Sends the telescope to the specified right ascension and declination, on the J2000 equinox.

Parameters

- **ra** (*float*) – the target right ascension, in degrees
- **dec** (*float*) – the target declination, in degrees

`get_target``()`

Returns The current target position of the telescope.

Return type CelestialPosition

`get_position``()`

Returns the current position of the telescope. The position is provided based on sidereal coordinates, regardless of the target position's characteristics.

Return type CelestialPosition

5.4 Dome Interface

5.4.1 Dome State Constants

class `ace.dome.`**`state`**

`AJAR`
Dome is neither open nor closed.

`CLOSED`
Dome is closed.

`CLOSING`
Dome is closing.

`ERROR`
An error condition exists.

`OPEN`
Dome is open.

`OPENING`
Dome is opening.

`SLEWING`
Dome is slewing to a position.

`TRACKING`
Dome is tracking a position.

5.4.2 Control Functions and Data

class `ace.dome.`**`Dome`**

`go_to`(*azimuth*, *elevation*, *block=True*)
Sends the dome to the specified azimuth and elevation.

Parameters

- **azimuth** – the desired dome azimuth

- **elevation** – the desired dome elevation
- **block** – causes the function to block while the dome is slewing

`open``()`
: Opens the dome.

`close``()`
: Closes the dome.

`stop``()`
: Stops all motion of the dome.

`azimuth`
: The current dome azimuth.

`cmd_azimuth`
: The azimuth where the dome has been commanded.

`cmd_elevation`
: The elevation where the dome has been commanded.

`park_azimuth`
: The azimuth of the dome when it is not in use.

`state`
: The current dome state.

5.5 Filter Wheel Interface

5.5.1 State Constants

class `ace.filterwheel.`**`state`**

`UNKNOWN`
: The wheel's state is unknown, or none of the other states apply

`STOPPED`
: The wheel is stopped at a valid position

`INITIALIZING`
: The wheel is initializing

`MOVING`
: The wheel is moving to a new position

`FAULT`
: The wheel is not in a valid state

5.5.2 Control Methods and Data

class `ace.filterwheel.`**`FilterWheel`**

`state`
Current state of the filter wheel, one of `state.UNKNOWN`, `state.STOPPED`, `state.INITIALIZING`, `state.MOVING`, or `state.FAULT`.

`go_to`(*pos*)
Sends the filter wheel to the specified position, identified by name or position number (starting from zero).

Parameters **pos** (*integer or string*) – filter wheel position to go to

`init`()
Re-initializes the filter wheel.

`stop`()
Stops motion of the filter wheel.

`position`
Current filter wheel position

`positions`
Total number of available positions on filter wheel

`filter_name`
Name of the current filter, or `---` if no filter is selected

`target`
Position that the filter wheel is attempting to move to

`get_names`()

Returns A list containing the name of each filter in the wheel.

Return type list of strings

`set_names`(*names*)
Sets the names associated with the filter positions

Parameters **names** (*list of strings*) – the filter names

`get_focus_offset`()

Returns The assigned focus offset for the current filter.

Return type float

`get_focus_offsets`()

Returns a list of the focus offsets for all the filters in the wheel.

Return type list of floats

`set_focus_offsets`(*offsets*)
Sets the focus offsets for each of the filters in the wheel

Parameters **offsets** (*list of floats*) – the new focus offsets

5.6 Focuser Interface

5.6.1 Focuser State Constants

class `ace.focuser.`**`state`**

`LIMIT_FWD`
The focuser is at the forward limit.

`LIMIT_REV`
The focuser is at the reverse limit.

`MOVING_FWD`
The focuser is moving in the forward direction.

`MOVING_REV`
The focuser is moving in the reverse direction.

`STOPPED`
The focuser is stopped between the forward and reverse limits.

5.6.2 Control Functions and Data

class `ace.focuser.`**`Focuser`**

`state`
Current focuser state.

`position`
Current position of the focuser.

`target`
The position that the focuser is moving *to*.

`minimum`
The minimum allowed focus target position.

`maximum`
The maximum allowed focus target position.

`go`(*pos*)
Sends the focuser to the specified position.

Parameters **pos** – the target focus position

`go_to_minimum`()
Sends the focuser to the reverse limit.

`go_to_maximum`()
Sends the focuser to the forward limit.

`go_to_preset`(*name*)
Sends the focuser to one of the preset positions.

Parameters **name** – the name of the preset to go to

`stop`()
Stops all focuser movement.

`set_preset`(*name*, *position*)
Sets a new preset, or overwrites an existing preset position with a new focus value.

Parameters

- **name** – name of the new preset position
- **position** – position value that should be associated with this name

`remove_preset`(*name*)
Removes an existing preset position.

Parameters **name** – name of preset to remove

`get_preset`(*name*)
Looks up the position associated with a preset.

Parameters **name** – name of the preset position to retrieve

Returns The value associated with the requested preset position

Return type float

`list_presets`()

Returns a list of the names of all available presets

Return type list of strings

5.7 Switch Interface

5.7.1 Control Functions

class `ace.switchable.`**`Switchable`**

`is_active``()`

Returns `True` if the switch is currently set to the "active" position.

Return type boolean

`set_active``(`*state*`)`

Changes the state of the switch

Parameters **state** (*boolean*) – target state for the switch

5.8 Example Python Script

The following script demonstrates the basic functionality of the Python scripting interface. The script makes a connection to `localhost` and uses a telescope, a camera, and a filter wheel.

It moves the telescope to each of three different celestial targets and takes a series of images in various filters, binned 2x2, with each filter having an associated exposure time.

```
import ace.syscore
import ace.camera
import ace.telescope
import ace.filterwheel

import astropy.coordinates as coord

conn = ace.syscore.AceConnection('localhost')

wheel = ace.filterwheel.FilterWheel(conn, 'Telescope PC', 'ACE Filter Wheel')
camera = ace.camera.Camera(conn, 'Telescope PC', 'Main Camera')
telescope = ace.telescope.Telescope(conn, 'Telescope PC', 'Telescope')

target_list = ( 'M37', 'NGC 3521', 'Eskimo Nebula' )
exposure_list = (
    ( 'U', 30 ),
    ( 'B', 20 ),
    ( 'V', 15 ),
    ( 'R', 10 ),
    ( 'I', 5 )
)
n_exposures = 5

for target_name in target_list:
    print 'Moving telescope to', target_name
    pos = coord.ICRS.from_name(target_name)
    telescope.go_to_j2000(pos.ra.degree, pos.dec.degree)

    for (filt, exptime) in exposure_list:
        print 'Moving filter wheel to:', filt
        wheel.go_to(filt)

        s = '%s_%s_{{seq}}.fits' % (target_name, filt)
        print 'Setting image filename template to:', s
        camera.template = s

        for i in xrange(n_exposures):
            camera.expose(exptime, bin_x=2, bin_y=2)

print 'Done.'
```

INDEX