April 2008

Automation of a Remote Telescope Imaging System

Jorge A. Alejandro
Worcester Polytechnic Institute

Muzhtaba Tawkeer Islam
Worcester Polytechnic Institute

Thomas Piotr Niemczycki
Worcester Polytechnic Institute

Automation of a Remote Telescope Imaging System

Project Report

A Major Qualifying Project
submitted to the faculty of
WORCESTER POLYTECHNIC INSTITUTE
and performed at SRI INTERNATIONAL
in partial fulfillment of the requirements for the
Degree of Bachelor of Science

Date: March 7, 2008

WPI Advisors:
Professor John A. Orr
Professor David Finkel

SRI Mentors:
Andrew Young
Elizabeth Kendall

Submitted by:
Jorge A. Alejandro
Muzhtaba T. Islam
Thomas P. Niemczycki

Table of Contents

Table of Figures

Table of Tables

Acknowledgements

We would like to express our sincere gratitude to the many people who made it possible for us to complete this project. We are thankful to professors John Orr and David Finkel of WPI who shared their knowledge and expertise to help us along. Thanks to their guidance, we were able to overcome the many technical challenges presented by this project. We would also like to express thanks to Andrew Young of SRI for being an excellent mentor and challenging us to develop a better project than we thought we could. We are also grateful to Elizabeth Kendall of SRI, who put forth extraordinary effort to arrange a visit to the HAARP facility in Alaska. Finally, we would like to thank the wonderful staff at SRI and HAARP who were extremely supportive throughout our project experience.

Abstract

SRI International is working with the High Frequency Active Auroral Research Project to develop instrumentation for researching the properties of the ionosphere. The project team developed a web-based interface that allows remote control of an optical imaging system located in Alaska. The interface provides control of the on-site cameras, optics, and mount for researchers at SRI. Data that is collected by the system is returned to SRI autonomously. The project concluded with on-site testing of the system.

Executive Summary

SRI International is working in conjunction with the High Frequency Active Auroral Research Program (HAARP) to develop instrumentation for studying properties of the ionosphere. As part of this research, a new optical telescope for observing ionospheric phenomena known as "airglow" was installed at HAARP's research facility outside of Gakona, Alaska. The telescope is used to make observations when sections of the ionosphere are excited by a high-frequency radio transmitter. In the former configuration, a researcher from SRI had to travel to Alaska and operate the telescope on-site in order to make observations. The team's project was to develop an automated remote control system that allows researchers control of the observatory equipment from California.

The telescope system is composed of two cameras, a set of optics for each camera, and a robotic mount that supports and orients the system. The camera and optics systems are identical, with the exception that one has a wide-angle lens and the other has a telescope lens. The cameras are attached to the same mount and always point in the same direction. The cameras, optics, and mount are controlled by two computers that are located on-site. The imaging system is housed inside an observatory dome that includes a slit for the telescope to be able to observe a portion of the sky. The motorized dome can be rotated so as to orient the slit in any direction.

The project team developed a web-based interface for the imaging system to allow remote control of both cameras, sets of optics, and the mount. The user interface is a website with fields that allow the user to enter settings such as desired mount orientation or camera exposure time. A user can also specify a time at which the observation session should start and stop. The web interface utilizes Javascript and XML files to update displays of the hardware status without refreshing the entire page. While an observation is taking place, new images appear on the page automatically as they are acquired by the cameras.

The web interface is hosted by an Apache HTTP server that is running on one of the on-site computers. The Apache server provides user authentication and SSL encryption to secure the system. When the user submits settings for the equipment, a script is called to process the submission. This script, which was written in the Python programming language, translates the settings into commands that are sent to the "back-end driver" over a TCP/IP connection. A back-end driver operates on each of the two computers that are connected to the hardware.

The back-end driver controls operation of the hardware on each of the two computers. This Windows console application is composed of several software classes that operate under the control of the "observation" class. When the observation class receives commands from the TCP/IP connection, it parses them and passes them along to the appropriate device class. The

device classes provide high-level function calls that cause the hardware to perform certain actions, such as acquire an image. These device classes interact with software interfaces that were provided by the manufacturers for each device. Each device class also reports back to the observation class, which uses the latest data from each device to generate an XML file. This XML file provides an updated hardware status for the website interface.

Images that are acquired by this system are stored in two formats. The WinView format appends a header to the raw pixels generated by the camera so that researchers may perform analysis on the data. Each image is also encoded in the Portable Network Graphics (PNG) format, which is compact and web-friendly. In order to see the airglow in the images, however, the values of pixel intensities are redistributed across the full range of grayscale colors available in the image. Once the image is encoded as a PNG it can be displayed on the user interface and made available for public viewing.

There are some additional features to make the system more robust. The WinView data files are automatically sent back to California as they are acquired by a program called DeltaCopy, which was installed on-site. A web-controlled power switch, which includes a built-in HTTP server, was installed on-site to allow users in California to switch devices on and off remotely. In order to provide remote maintenance, the on-site computers were configured to allow Virtual Network Computing (VNC) access, which allows remote users to control the desktop of each computer. Additionally, the back-end driver includes telnet access through its TCP/IP module, which allows a user to debug individual commands.

One component that was not implemented is automation of the dome that houses the imaging equipment, which was an original goal of the project. Upon arriving at SRI, the team learned that there is no existing interface between the dome and a PC. Additionally, accounts of past failures of dome components raised concerns over operating the dome remotely. However, most of the components required for automated dome operation are already in place. The missing components are two position sensors and a PC interface. The product team developed a proposal of the equipment that would be necessary to complete dome automation, which would provide remote control of the entire observatory. By following the team's recommendations, the dome can be operated safely from a remote location.

The project has been tested extensively on the equipment to be as reliable as possible. Near the end of the project, the team travelled to the HAARP facility in Alaska to deploy and debug the system. By the end of the trip, a researcher was able to control the imaging system from SRI in California. The project has provided a feasible way to control the equipment remotely and collect image data.

1. Introduction

SRI International is working in conjunction with the High Frequency Active Auroral Research Program (HAARP) to develop instrumentation for studying properties of the ionosphere. As part of this research, a new optical telescope for observing ionospheric phenomena was installed at HAARP's research facility outside of Gakona, Alaska. This telescope is used to make observations when sections of the ionosphere are excited by a high-frequency radio transmitter. In the former configuration, a researcher from SRI had to travel to Alaska and operate the telescope on-site in order to make observations. With the completion of this project, SRI has an automated system that can remotely control the telescope's operation from the company's facility in Menlo Park, California.

The team's project was the development of a system to automate operation of the optical telescope. The solution allows for autonomous remote operation of the cameras, optics, and mount that make up the telescope system. The remote system allows the observatory to remain fully functional while eliminating the need to operate the imaging equipment on-site.

2. Background

SRI International is a nonprofit research institute that conducts client-sponsored research and development. The organization's mission statement stresses its commitment to discovery and the application of science and technology for development of knowledge, commerce, prosperity, and peace. Its list of clients includes government agencies, commercial businesses, foundations, industry consortia, and venture firms. SRI employs about 1,400 staff members and its 63-acre main campus is located in Menlo Park, California.

Originally founded in 1946 by the trustees of Stanford University as the Stanford Research Institute, the company became independent and changed its name to SRI International in the 1970s. Over the course of its history, SRI has developed a legacy of innovation in a wide array of fields including communications and networks, economic development, health sciences, the environment, and materials and structures. The company also brings its innovations to the marketplace by licensing its intellectual property, and creating over a dozen new spin-off ventures in the process. SRI's for-profit subsidiary, Sarnoff Corporation, led the development of the High Definition Television standard and has won 10 Emmy awards for outstanding achievement in technological advancement. Other notable innovations from SRI include the computer mouse, the first computer network known as the ARPANET, optical read-write data storage, the liquid crystal display, the next-generation heart valve, and breakthroughs in surgical robotics.

With more than sixty years of experience in developing technology, SRI is looking forward to tackling some of the world's most pressing problems. Over the past year, SRI has made numerous advancements in medicine by continuing to refine its robotic surgeons, improving diagnostic tools for cancer, and developing technologies for rapid manufacturing of vaccines and drugs. SRI is also conducting geospace research, which includes ongoing studies of the polar atmosphere taking place in Greenland and Alaska.

2.1. High Frequency Active Aural Research Program

The High Frequency Active Auroral Research Program (HAARP) is a project aimed at gaining a better understanding the ionosphere for the purpose of developing more advanced communication and surveillance systems. Started in 1993 and proposed to last for twenty years, the project is funded by the United States Air Force, the United States Navy, and the University of Alaska. The project site is located near Gakona, Alaska, and consists of two main components. One main component is an Ionosphere Research Instrument (IRI) - a high power, high frequency transmitter that is used to excite the ionosphere . The second main component is an array of scientific instruments that are used to study the responses to ionosphereic disturbances. The latest IRI at the site has a 3,600kW transmitter and 15x12 antenna

arrangement. The continually expanding facility is constructing new equipment to advance its research.

Recent developments at HAARP include the UHF ionospheric radar, which is a 446MHz phased array of 512 antenna elements. This radar, which is used to study the physical processes of the ionosphere during operation of the IRI, is only part of a larger design known as AMISR (Advanced Modular Incoherent Scatter Radar). AMISR is being developed by the National Science Foundation (NSF) and SRI International. During the final implementation stage, a new computer-controlled optical telescope has been installed inside a 14 foot diameter dome. With the addition of these and other new instruments, HAARP continues to pioneer study of the ionosphere.

2.2. Ionospheric Research

The ionosphere is a layer in the Earth's atmosphere that is located above the troposphere and the stratosphere. Due to the thin air that exists in this region of the atmosphere, solar radiation strikes the gas with enough intensity to separate electrons from neutral molecules. These collisions produce free electrons and positively charged ions, both of which exist for a short amount of time. A high-power signal may be transmitted directly to the ionosphere, further exciting the molecules to produce a phenomenon known as "airglow." The airglow may then be studied from earth using antenna arrays and optical imaging devices. The data gathered from these studies provides insight into the composition and behavior of the ionosphere.

Ionospheric research has profound implications in communications and navigation systems. For example, broadcasters utilize the properties of the ionosphere to reflect radio signals around the curve of the Earth so that their programs can be transmitted across the globe. The Voice of America (VOA) and the British Broadcasting Corporation (BBC) are two such international broadcasters. The ionosphere also provides for communications between ships and the shore, for trans-oceanic aircraft communication, and for military surveillance systems. These useful applications provide reasons for more advanced research into auroral activity and the properties and composition of the ionosphere.

3. Technical Backround

The telescope system is used to conduct observations of airglow that results from operation of the HAARP transmitter. The telescope system is composed of several pieces of equipment that are used together when conducting an observation. These devices are a telescope mount, two cameras, a set of optics for each camera, and two personal computers (PCs). The system components can be configured for various applications, but the way in which their features are used at HAARP influenced the design of this project. This technical background section describes each of the components and explains how they are used in practice.

3.1. Telescope System and Operation

The telescope system is arranged as shown in Figure 1 on the next page. Both cameras, one with a telescope lens and the other with a wide-angle lens, sit on top of the telescope mount. The mount moves about two axes; it can rotate to point the cameras in any direction, and it can move from a horizontal to a vertical position to orient the cameras toward the sky at a given angle. Both cameras always point in the same direction. The two cameras are identical with the exception that one is attached to a telescope, which provides a close-up image with a narrow field of view, and the other one is attached to a wide-angle lens, which offers a distant image with a wider field of view. Attached to each camera is a set of optics. Each set of optics includes a filter wheel that is used to select a filter for the image, and an external shutter. This shutter sits at the camera lens and can be opened or closed to allow light to enter or to keep light and debris out. Two PCs are used to control the telescope system. The telescope camera and telescope optics are attached to one PC. The wide-angle camera, its optics, and the mount are attached to the other PC. In practice, both cameras are used simultaneously to take close-up and context images of HAARP airglow.

The imaging system is used to conduct observations while a HAARP experiment is in progress. The experiments are scheduled to start and stop at predefined times given in Universal Coordinated Time (UTC). During each experiment, the HAARP transmitters operate in on and off intervals of a given length. While running, they transmit signals into the ionosphere in an attempt to create airglow, which is in turn detected by the optical imagers. The transmitters are typically aimed so as to produce airglow along the magnetic field lines of the earth or directly above the facility. The cameras are aimed accordingly by positioning the mount.

While an experiment is occurring, a researcher specifies an exposure time, filter, and a data readout rate. These settings are explained in detail in section 3.3. Once the cameras are set up, they are started simultaneously and take images continuously. When the experiment is complete, the raw image data is saved to a hard drive or CD for later analysis.

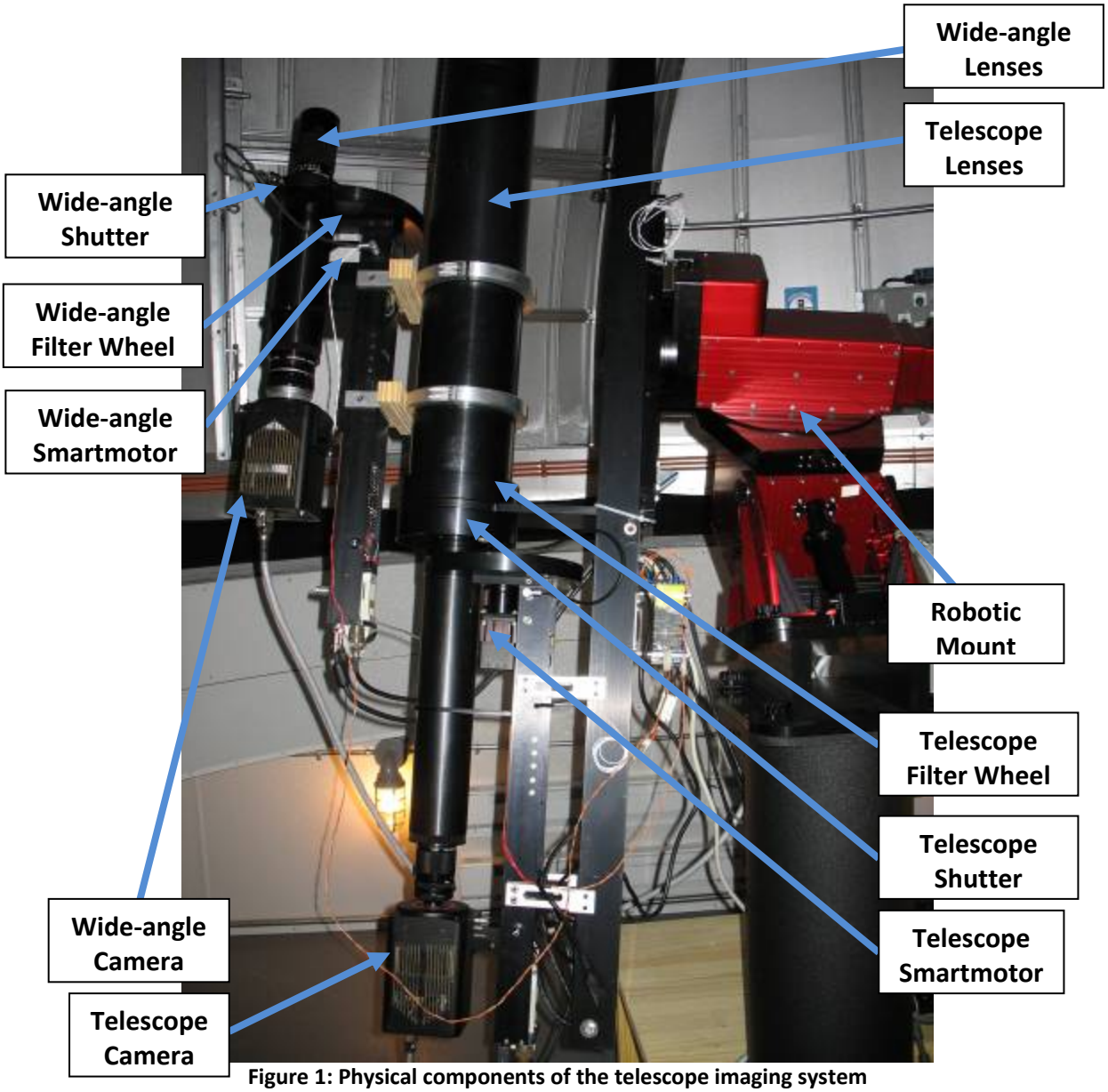


Figure 1: Physical components of the telescope imaging system

3.2. Optics

Two sets of optics with almost identical interfaces are in use at the HAARP site. Although each set has several components, we are only concerned with the active components: the external shutter, the filter wheel, and the SMARTMOTOR system controlling them.

The external shutter sits just outside the camera. Its main purpose is to protect the sensitive internal circuitry of the camera from exposure to bright light. The shutter has two positions, opened and closed, and it is controlled directly by the SMARTMOTOR system. The SMARTMOTOR system has three inputs for controlling the shutter: a manual switch located on the optics circuitry, an external trigger input located near the shutter, and an RS-232 interface used to communicate with the SMARTMOTOR from a PC. The project uses only the RS-232 interface to control the shutter.

The filter wheel is a disc that contains six circular openings along its edge. Each opening houses a distinct filter. The filter wheel is mounted such that as it rotates about its center point, each of the filters covers the camera opening individually at one point in time. The SMARTMOTOR controls this rotation and also ensures that the filter wheel is never "stuck" between filters. Filter wheel control can only be carried out using the RS-232 link. The commands used to control both the filter wheel and optics are listed in section 6.5.1.

3.3. Cameras

There are two identical charge-coupled device (CCD) cameras in use at the HAARP site. A CCD camera functions by collecting light and converting it into charge and then emitting the signal that results in a digital image. The model of the on-site cameras is Princeton Instruments VersArray 512B, and they are manufactured by a company called Roper Scientific. This specific model has been phased out and is no longer in production.

The cameras are connected to controllers, which in turn interface with the PCs via PCI cards. Each camera has its own controller (model ST-133), which sets the hardware temperature to -40^{O}F on power up. Such a low temperature is necessary in order to minimize the effects of thermal noise on the image quality. A higher temperature generally results in a grainy image due to the higher background noise levels. The cameras come with their own software package, WinView, which has some built in image processing capabilities. Winview supports its own image format; WinView files have the extension .SPE. Figure 3 shows the two cameras installed on site, and Figure 3 shows the two controllers.

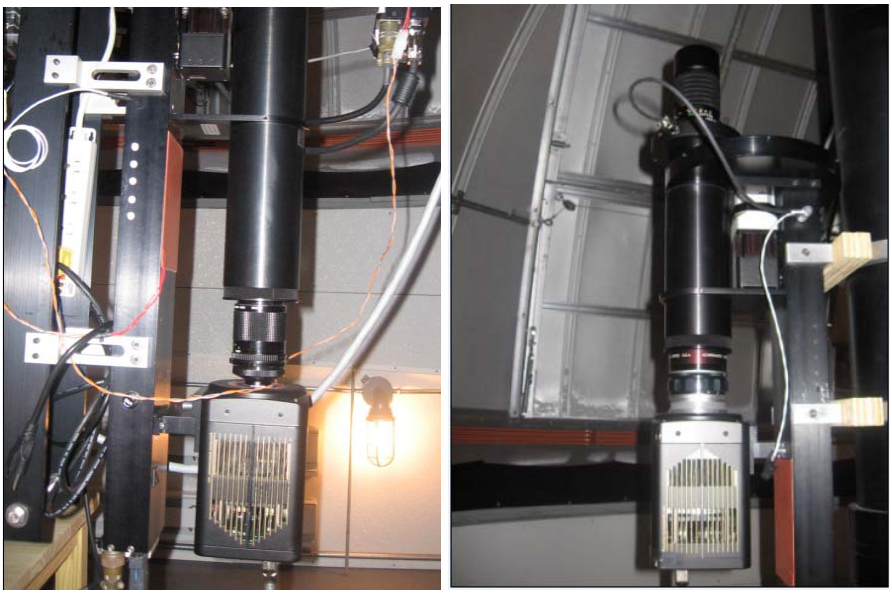


Figure 2: Telescope (left) and Wide-angle (right) cameras

Figure 3: ST-133 controllers for the cameras

The cameras from Princeton Instruments are supported by PVCAM (Programmable Virtual Camera Access Method Library). PVCAM is an ANSI C library of camera control and data acquisition functions for cameras from Roper Scientific. The library is independent of different platforms and operating systems. It provides a high-level interface to the camera device driver so that developers may choose to specify the camera's setup, exposure, and data storage attributes. In our design approach, we made a strategic decision to implement the camera functionality via the PVCAM library, as opposed to using other imaging software. This decision is discussed in section 5.

Figure 4 is a pictorial view of the different layers that interact through PVCAM.

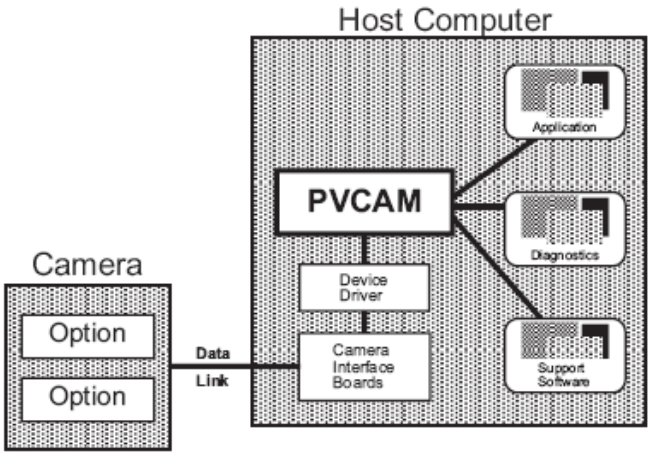


Figure 4: PVCAM Interface
(source: PVCAM Manual)

The PVCAM library allows direct access to the low-level device drivers for the camera module. The WinView software is an application program that interfaces with PVCAM to control the camera hardware. One important image processing feature included in WinView is the "5-95" algorithm. This algorithm allows the user to rescale the image pixels to make faint features more visible. This operation allows the airglow to show up in the images once the re-scaling process is applied.

3.4. Mount

The mount that is located at the HAARP site is a Paramount ME Robotic Mount manufactured by Software Bisque. The mount physically supports both sets of cameras, camera lenses, and optics. The Paramount ME moves about the vertical and horizontal axes, and its operation is controlled by a computer. The mount connects via USB to the same PC that also controls the wide-angle camera and optics.

On the PC, mount operation is controlled through a program called TheSky6. TheSky6 is a robust software package that gives the user a lot of functionality in addition to controlling the mount, such as tracking specific celestial objects across the sky. In practice at HAARP, however, the software is only used to perform basic functions, such as orienting the mount. Since the control protocol for the Paramount ME is proprietary and has not been released by the manufacturer, using TheSky6 is currently the only way to control the mount with a PC.

After TheSky6 is started on the PC, the program can connect to the mount to control it. The user can then orient the mount by slewing it to a given position. In order to slew the mount, the user specifies a desired position in terms of azimuth and altitude. Azimuth specifies the position around the horizon; an azimuth of 0° indicates north, 90° is east, 180° is south, and 270° is west.

Altitude specifies the height above the horizon; the horizon has an altitude of 0°, and straight up is indicated by an altitude of 90°. Both azimuth and altitude are specified in terms of degrees, minutes, and seconds, where sixty seconds is one minute and sixty minutes is one degree.

The mount can be initialized to an absolute position through a procedure called homing. Homing the mount moves it to a home position, which is a fixed mechanical orientation that is defined by the orientation of the gears relative to internal homing sensors. When the mount reaches the home position, its position registers are reset to zero. All subsequent movement can be performed accurately relative to that accurate position. The mount must be homed whenever it is turned on, and it cannot operate until it has been homed. Additionally, homing allows the mount to recover to an accurate position in the event of a power outage or other interruption.

Another operation that the user can perform is called parking, which places the mount in a certain position when it will not be used. The mount is parked to a predefined position, which is set by the user. In practice, this position is one that puts the telescope equipment in a convenient position for people to move about the limited space in the telescope enclosure.

At HAARP, the mount is typically operated in only a limited portion of its complete range of motion. The orientation for viewing airglow directly overhead is the same as the park position. The orientation for observing at the earth's magnetic zenith, currently 202°0'0" azimuth 76°0'0" altitude, is very close to the home position. In moving between all these positions, the mount only rotates slightly about both axes. There is no risk in moving between these positions without observing the mount. All cables connected to the instruments were tied off so as to minimize the risk of becoming tangled around the mount during a slew. A user can slew the mount outside of the normal range of operations safely, but it may be beneficial to have personnel on site to ensure that the cables do not become tangled as the mount moves to that position.

3.5. Optics Shelter

The telescope system resides in an enclosure known as the optics shelter, which is located about a quarter mile from the HAARP transmitter. The shelter is a small booth with two rooms, one to house the telescope and the other for computer equipment. The telescope enclosure is outfitted with a fourteen-foot diameter dome manufactured by Ashdome. The dome is a steel hemisphere formed by a number of interlocking segments with a shutter system that opens a slit on one side of the hemisphere. The slit allows the telescope to view any portion of the sky while protecting it from an unnecessary exposure to wind and other weather conditions. The optics shelter is shown in Figure 5.

Figure 5: Optics Shelter and Dome

As the telescope cannot focus through glass, it must be exposed through the dome slit in order to make observations. During observation periods, the cover swings back to open the slit, allowing the telescope to gaze at the sky. The lower portion of the slit is opened by lowering a drop-out door. Once the slit is opened, the entire dome structure can be rotated, allowing the slit to be aligned with the telescope so that it may peer at any portion of the sky. The slit cover can be closed to protect the telescope from damage due to rain or snow.

Currently, the dome can be only controlled manually on-site, and there is no PC interface to its controls. Possible methods for enabling autonomous and remote operation of the dome are discussed in Section 8 of this report.

4. Project Goals

The goal of this project was to develop a system for automated and remote control of a telescope that is operated in conjunction with the high frequency radar instrumentation at the HAARP facility in Alaska. The remote system must provide users at the SRI facility in California the capability to control all of the components that make up the telescope system.

The internet is an obvious two-way communication medium for remote control of the telescope system as a high-speed connection is available right inside the optics shelter. Additionally, the cameras, optics, and mount are controlled by PCs that are already connected to the internet. Therefore, the overarching technical challenge presented by this project was to extend system control from the on-site computer terminal to a remote end of the internet connection. Figure 6 shows a conceptual overview of the system.

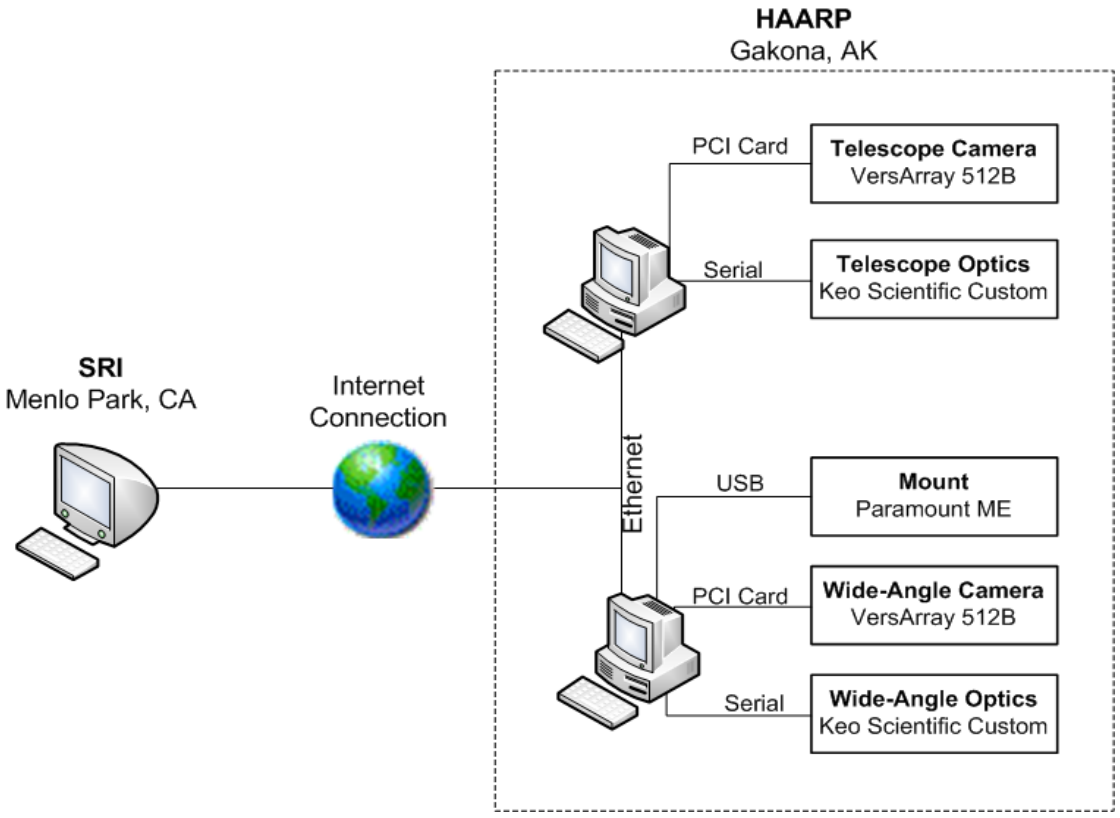


Figure 6: Remote Telescope System Overview

Software for controlling the individual components of the imaging system exists, but each device is controlled by a separate software package. Prior to this project, these software packages have never been put together to work in a synchronized manner. The solution required an integrated interface that allowed remote control of all the functions that are executed by these software packages.

The project team consulted with Elizabeth Kendall of SRI, the primary user of the telescope system, regarding any preferences for the user interface. Elizabeth specified that she uses the telescope and wide-angle imagers simultaneously, and that she would like to control both at the same time. She also specified that she needed to be able to set the exposure time, analog-to-digital (ADC) conversion rate, and filter status for each observation, and that she needed remote control of the external shutter on both imagers. Other camera settings, such as camera temperature, could be preset as they would not need to be adjusted by the user. Data produced by each imager must be stored in a raw format, as well as a web-friendly format for posting to a website. Finally, Elizabeth needed to be able to move the mount to a given azimuth and altitude. The ability to open and close the dome slit and keep it aligned with the telescope and the ability to detect hazardous weather conditions were desired as well.

Aside from the explicit requirements provided by Elizabeth, the nature of the project also presented some implicit requirements. As this system operates at a remote site, it needed to be reliable and offer remote maintenance capability. Additionally, the solution needed to be flexible enough so as to allow the user to continue being able to utilize existing software should the need arise. That is, the automation system should not reduce functionality, but expand on the capabilities already in place.

5. Evaluation of Design Options

Early research into possible solutions presented several paths for the project and a number of ways to potentially implement the desired functionality. The project team evaluated the advantages and disadvantages of each solution.

The simplest solution was to provide access to the remote computers through a Virtual Network Computing (VNC) connection. A VNC server was installed on each of the on-site computers. By using the VNC client, a user at SRI can see the desktop of each on-site computer and control it with the local mouse and keyboard. This method allowed for the use of the existing software to control the hardware remotely. This solution, while simple, did not automate the operation of the imagers. However, the team left the VNC capability in place because it allows previously installed software to be used remotely, and it provides an easy way to troubleshoot problems with the system and automation software.

An original design for the system relied on the existing remote control functionalities of astronomy software from Software Bisque, the same company that manufactured the telescope mount. While the company produces imaging software with remote access features, called CCDsoft, the product does not include built-in support for the cameras and optics that are installed at HAARP. Attempts to develop a custom driver to operate the cameras and optics with CCDsoft showed that it would be impossible to incorporate some settings that are specific to these cameras into the remote interface of this software package. Since this path would not allow us to meet the user's requirements, we decided to abandon it and proceed with another approach.

An alternate solution was to develop custom software to control the hardware. The remote interface was to be a client-side application, which would connect to a server-side application that ran on the on-site computers. Ultimately, we moved away from the client-side application in favor of a website interface, which was suggested by Andrew Young of SRI. A client side interface would require extensive programming. A website, however, is simple to set up and allows text and images to be arranged on the screen easily. Additionally, a website allows easy portability from one remote computer to another. The next section discusses the implementation of this design and provides information about each of the software components.

6. Remote Telescope Control

The remote telescope system enables a user at SRI in California to control the cameras, optics, and mount located at the HAARP facility in Alaska. The following sections detail each of the software components of the system. A user's guide to the system is provided in Appendix A. The maintenance manual for the system is available in Appendix B.

6.1. Design Overview

The remote telescope control system provides the user with a web-driven interface, which interacts with a software component that runs on the on-site computers. Between the user and the hardware lie several layers of software, each with a specific role in the system. Figure 7 provides an overview of these major software components.

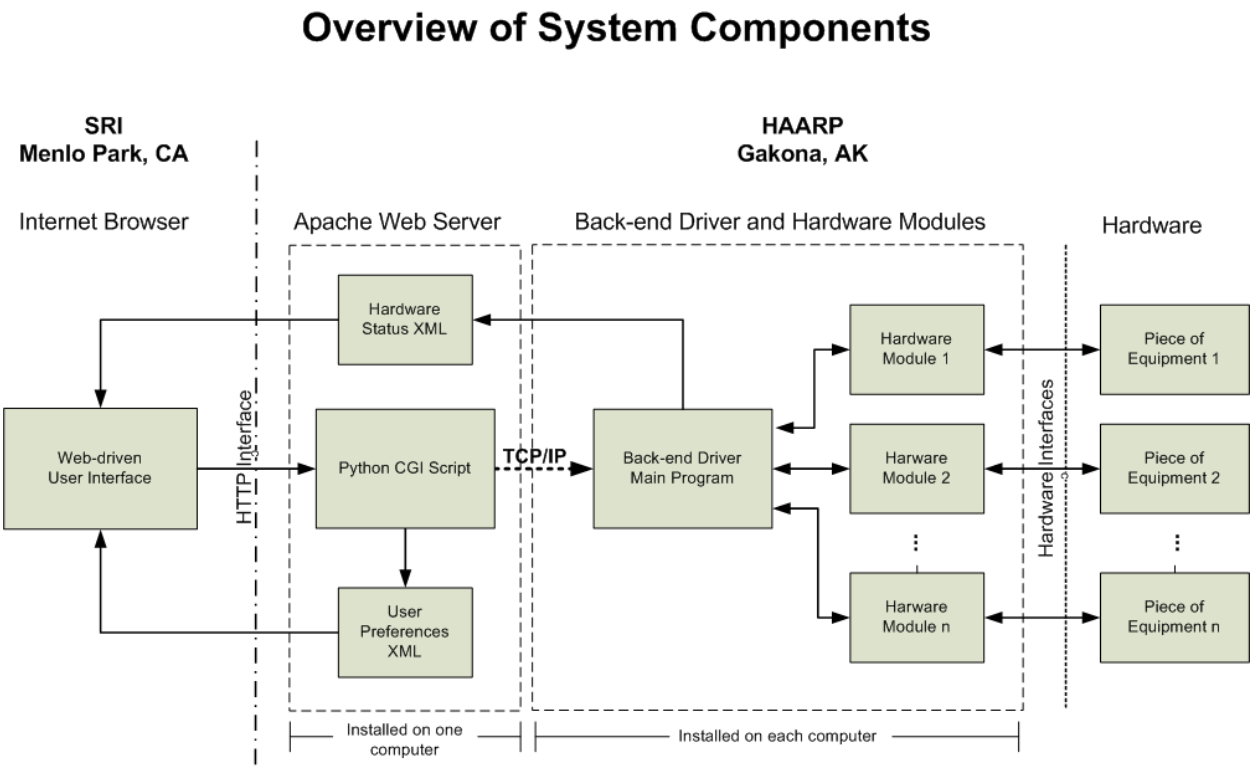


Figure 7: Overview of major system components in the remote telescope system

On the left side of the figure is the user interface, which can be opened in any web browser at SRI. The web browser connects over the internet to the Apache web server, which is running on-site. For increased security, the HAARP firewall only allows the user interface to be accessed from an IP address within the block of SRI addresses. The web server receives incoming connections from web browsers and sends back web pages, XML data files, and images over the Hyper Text Transfer Protocol (HTTP). The web server can also receive user inputs, which it then passes along to a back-end driver using Transmission Control Protocol/Internet Protocol (TCP/IP) over the on-site local area network.

There is a back-end driver installed and running on each of the two on-site computers that are connected to hardware. This program is the heart of the system as it listens for commands, autonomously controls the hardware, and reports the status of each piece of equipment. Finally, the hardware modules are the interface between the back-end driver and the hardware itself. These modules are libraries of functions that are built into the back-end driver. They are specific to the hardware, and the back-end driver uses these modules to operate each piece of equipment.

The remote telescope control system runs under the Microsoft Windows XP operating system. Roper Scientific PVCAM version 2.7 must be installed in order to start the program, and Software Bisque TheSky6 must be installed in order to use any of the mount functions. In order to provide a web-driven interface, Apache Server version 2.2.8 or later should be installed and configured according to section 6.3.1**.** Additionally, the Python programming language, version 2.5 needs to be installed in order to run the included CGI script, which is discussed in section 6.3.2.

Information is sent to the back-end driver from the web server over TCP/IP port 5000. Therefore, computers that operate as part of the remote telescope system must be able to communicate over a local network using that port.

The computers must also be able to share a network folder. The back-end driver generates files locally, in the directory that it is placed. In order for these files to be accessible to a web interface, a back-end driver that is running on a computer other than the web server should be placed in a network drive that maps to the server. In our installation of the system, the computer connected to the wide-angle camera hosts the Apache web server. Both backend driver executable files actually reside on this computer, but one of them is placed in a shared folder that is mapped from the other computer over the local network. Running the back-end driver on the telescope computer from this shared network folder allows it to generate files that are visible to the other computer as well. Therefore, both computers must be connected to the network to use the telescope camera and optics through the remote telescope system. A list of all the files and their directory locations is provided in Appendix C.

The following sections discuss each of the major components of the remote telescope system.

6.2. Web Driven User Interface

A user controls the remote telescope system through a website interface. The web interface was developed for Microsoft Internet Explorer 7 and Mozilla Firefox (version 2.0), and the user should experience the same functionality using either browser. Note, however, that the project team observed occasional and irregular delays in updating the data displayed on the page when using Firefox.

The user interface is accessed through a web browser by entering the IP address of the on-site host computer. The IP address must be preceded by *https://* and followed by colon and the port number, as the server operates over secure hyper-text transfer protocol on a non-standard port for additional security. Upon entering the server address, the user is immediately prompted for a username and password, which have been supplied to SRI personnel. Once logged in, the user is able to access all functions of the remote telescope system. In order for all the features of the user interface to work properly, the user must set browser settings to allow ActiveX controls and disable any pop-up blockers.

A navigation bar, which appears across the top of all pages, allows the user to quickly jump between all the pages of the website. The links in the navigation bar are "Mount Control," "Observation Control," and "Web Power Switch." A link to the HAARP Operations page, which is convenient for researchers when performing experiments, is provided as well and opens in a new browser tab.

6.2.1. Web Page Design with HTML and Javascript

Each of the web pages that make up the user interface is composed of HTML (Hyper-Text Markup Language) code, which dictates the layout of elements on the page, as well as Javascript, which allows content to change dynamically. A page corresponds to a single HTML file, which may then reference any number of Javascript or image files.

When the user first logs into the web server, the page *index.html* is automatically loaded. This page does not display any actual content, instead it sets the website title, "SRI Telescope Automation Project," and instructs the browser to display content in two frames. One frame is located across the top of the browser window and displays the navigation bar, which is actually an HTML page itself. The second frame fills the remainder of the browser window, and it displays the user interface. Clicking on "Observation Settings" or "Mount Settings" in the top frame will result in the respective page appearing in the main frame. Clicking the link labeled "Web Power Switch" will cause the interface of the web-power switch, which originates from the device's built-in server, to appear in a new window.

The Observation and Mount Control pages serve two primary purposes: to display the current status of the hardware and to provide a user-friendly interface for sending commands to the equipment. The latter is achieved through the use of web forms. Each page can contain one or more forms, and each form can contain any combination of text input fields, radio buttons, checkboxes, hidden fields, and buttons. A form is submitted by pressing a button on it, which triggers the browser to send the values of all the fields contained in the form to the server for processing. In addition to fields that allow the user to select a preference or enter information, a form can contain hidden fields that contain static values. These fields are submitted along

with the form, but the user never sees them on the webpage. In the mount and observation pages, hidden fields are used to identify which piece of equipment the user is trying to control. Submitting a form requires the entire page to be refreshed. Every form in the user interface is configured to return the user to the same page from which the form was submitted.

The key to dynamically updating the webpage to display the current status of the hardware is HTML div tags. Each tag is labeled with a unique ID, and it can be placed anywhere on the page. When the page is loaded in the user's browser, a Javascript replaces these div tags with additional HTML. Since the Javascript continuously runs inside the browser whenever the page is open, the content of each div tag can be updated and changed. Div tags can contain a simple value, or they can cause text and input fields to appear and disappear.

The entire user interface is composed of many files, each of which has a specific role. Table 1 summarizes all the files that are involved and provides a brief description of each.

Filename	Description
layout.css	Sets a consistent look for forms and text across all HTML pages
index.html	First HTML page to be loaded when browser accesses the server. It sets up the dimensions of the frames.
mount.html	Defines the mount control page
nav.html	Loaded into the top frame to display navigation links
observation.html	Defines observation control page
obsTelePic.html	Displays last telescope image full-size
obsWidePic.html	Displays last wide-angle image full-size
teleCamLastImg.png	The last telescope image in PNG format
wideCamLastImg.png	The last wide-angle image in PNG format
imgBig.js	Contains Javascript functions that download new status XML files and refresh the image if a new image was been taken. Includes functions for both telescope and wide-angle images.
mount.js	Contains Javascript functions to display Mount status dynamically.
observation.js	Contains Javascript functions to control Observation page.
mntUserPrefs.xml	Stores the coordinates the user entered when slewing telescope
mntStatus.xml	Written by the back-end driver. Stores the current position and status of the mount.
obsUserPrefs.xml	Stores the user's settings from when an observation was started.
telecamStatus.xml	Written by the back-end driver to store the status of the telescope camera and optics.
wacamStatus.xml	Written by the back-end driver to store the status of the wide-angle camera and optics.

Table 1: Summary of files that make up the web-driven user interface

The details of how these files work together are explained in the following sections. The code for each of the HTML and Javascript files that make up the user interface can be found in Appendix D. The next section describes how Javascript loads and processes data from the XML files.

6.2.2. XML Data Storage and Javascript Processing

XML is a standard for storing information in a text file. The use of XML files as a means of storing information about the current state of the hardware and user preferences provides a simple way to write and access the information asynchronously. When the user loads a page, the information has already been written and stored in the XML file. After the page is loaded, the browser can poll the server periodically to check whether new information was written to the file. At the same time, the back-end driver can write the data to the XML file at any time, and the file will contain the newest information the next time it is polled by the browser.

An XML file stores data inside of tags, within which may be found a value or additional tags. The names of the tags are arbitrary and can be defined by the user, but matching open and close tags must be present for each piece of data. Examples of each of the XML files used in the web interface can be found in Appendix E. The following example shows how data can be arranged in an XML file:

```
<dataset>
      <groupA>
            <parameter1>value1</parameter1>
            <parameter2>value2</parameter2>
      </groupA>
      <groupB>
            <parameter1>value3</parameter1>
            <parameter2>value4</parameter2>
      </groupB>
</dataset>
```

The reading and processing of information stored in an XML file is handled by Javascript. In this example, Javascript can extract the value of *parameter1* with the following syntax (note that in actual code, a Javascript command must appear on a single line):

```
Var parm1Val =
xmlDoc.getElementsByTagName("groupA")[0].getElementsByTagName("parameter1")
[0].childNodes[0].nodeValue
```

In this example, *xmlDoc* represents a file handle for the XML document that was set when the XML file was loaded. If *groupA* included additional sub-tags, they could be accessed by adding *getElementsByTagName* as shown above until the value was reached. Similar code appears in all the Javascript files used in the web interface.

The mount page and observation page each have a corresponding Javascript file that stores functions for processing XML information for the respective page. The observation and mount pages both poll XML files once every 3.5 seconds. Once the HTML content of each page is completely loaded, a Javascript function called *loadXML* is called automatically. This behavior is achieved by specifying the Javascript for each page in the "head" section of the HTML file, and then indicating "onload=loadXML" in the tag denoting the start of the "body" of the page.

In each page, the *loadXML* function downloads the appropriate XML file and assigns it a file handle for calling data from that file. The function then calls another function to process the file. The second function reads the fields of the XML file. Through the use of if-else statements, the Javascript can display one piece of HTML code in a given div tag on the page if the corresponding XML field contains true, or another piece of code if the field is false. XML fields can also store string and numeric information such as the time the file was updated or which filter is selected.

After processing the hardware status information in an XML file, the Javascript calls another function, called *controlField* in both the observation and mount Javascript files, to update the user control fields. Through the div tags, the Javascript can allow input fields to show on the page, allowing the user to enter commands, or hide them so that that the user cannot enter new commands before a previous operation has finished. If the fields are to be blocked, the Javascript displays what settings the user entered prior to starting the operation. This information is stored in a separate XML file, which is downloaded only if a piece of equipment has started an operation. It would be impractical to update the control fields more frequently because all the fields would reset every few seconds as the XML file is polled. Therefore, the control fields are updated only when it is necessary to allow or disallow the user form entering settings.

After all information and control fields have been updated, the last command to be executed by the Javascript on each page is the *settimeout* function. This function takes in the name of a function to load and the amount of time in milliseconds to wait before calling it. In the case of the user interfaces, the *settimeout* function tells the Javascript to wait 3500 milliseconds and then start from the beginning by executing the *loadXML* function. In this way, the Javascript continues to run inside the user's browser to download updated XML files, process the information they contain, and display data on the web page.

The subsequent sections explain how the mount and observation pages utilize XML fields and div tags, as explained above, to provide a dynamic user interface.

6.2.3. Mount Control Interface

The Mount Control page allows control over the mount operation. The page provides information about the status of the mount and allows the user to control its functions.
The top portion of the page displays information about the current status of the mount. This information comes from the back-end driver, which polls the hardware and monitors the status. The website refreshes the status display at regular intervals. The possible states of the mount that can be displayed are: *Connected, Disconnected, Slewing,* and *Homing*. Whenever the mount is connected and idle, the status will indicate *Connected*. The status changes whenever the mount begins to slew or home, and then reverts to *Connected* once the operation is complete. The page provides the user with the current position of the mount below the status. If the mount is disconnected, this portion will display “n/a” in each field. Otherwise, the current azimuth and altitude will display in degrees, minutes, and seconds. A screenshot of the mount interface is provided in Figure 8.

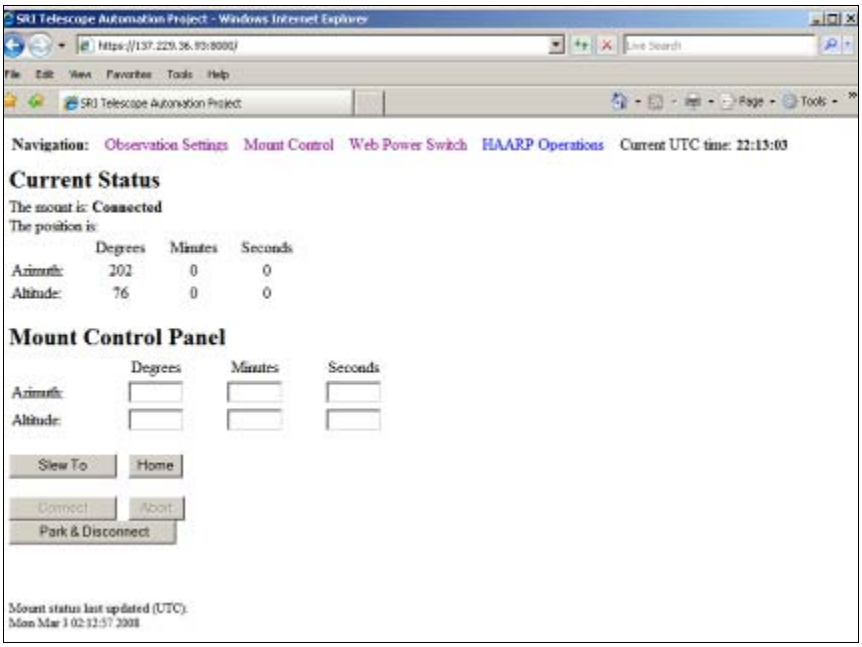


Figure 8: Mount Control Interface

Beneath the *Current Status* section of the mount control page appears the *Mount Control Panel* section. This section provides fields for the user to enter the desired azimuth and altitude and buttons to perform mount functions. The fields and buttons can be enabled or disabled, depending the current state of the mount. Table 2 summarizes what the control panel will display for each state.

Mount Status	Azimuth & Altitude Text	Slew To Button	Home Button	Connect Button	Abort Button	Park & Connect Button
Connected	User input box	Enabled	Enabled	Disabled	Disabled	Enabled
Disconnected	"n/a"	Disabled	Disabled	Enabled	Disabled	Disabled
Slewing	User's input	Disabled	Disabled	Disabled	Enabled	Disabled
Homing	"n/a"	Disabled	Disabled	Disabled	Enabled	Disabled

Table 2: Summary of available mount controls depending on state of mount

Enabling and disabling specific functions depending on the state of the hardware prevents the user from entering new commands before previously entered commands have finished executing. This behavior is achieved with the use of div tags, as described earlier.

Each button in the control panel serves a specific function. The *Connect* button instructs the back-end driver to connect to the mount, which allows the mount to process other commands. The *Home* button instructs the mount to move to the home position. Note that this process may take as long as a minute to complete. The *Abort* button can be pressed any time that the mount is moving and will cause the mount to stop immediately. The *Park & Disconnect Button* instructs the mount to move to the preset park position and the back-end driver to disconnect from the mount. The *Slew To* button instructs the mount to move to the azimuth and altitude specified in the input boxes located above the buttons. While the mount is in its connected state, the user can input values of 0-359 degrees for azimuth and 0-90 degrees for altitude. Both sets of minutes and seconds boxes take values of 0-59. The page will notify the user with a message box if invalid values or characters are entered into the user input boxes. Table 3 summarizes the fields that are sent to the server when the user submits the form, and which div tag displays each field.

Form Field Name	Field Type	Value	Div id
Equipment	hidden	mount	none
mntOp	hidden	slew	none
mntAziDeg	text	0-359	setAziDegDisp
mntAziMin	text	0-59	setAziMinDisp
mntAziSec	text	0-59	setAziSecDisp
mntAltDeg	text	0-89	setAltDegDisp
mntAltMin	text	0-59	setAltMinDisp
mntAltSec	text	0-59	setAltSecDisp
submit	submit	Slew To	none

Table 3: Fields submitted to server when slewing the mount

Each of the buttons at the bottom of the mount control page belongs to an individual form. Every form on this page includes the field named “equipment,” which specifies that these are mount commands, as well as a field called “mntOp” that specifies which mount operation is to be performed when the user presses the button on the form. These pseudo commands are intercepted by the Python CGI script and re-sent to the backend driver as the actual commands. The functions of the forms containing the buttons other than “Slew To” are listed in Table 4.

Button	“mntOp” Value	Backend Driver Command
Slew To	slew	mntSlewTo=TRUE
Home	home	mntHome=TRUE
Connect	connect	mntConnect=TRUE
Abort	abort	mntAbort=TRUE
Park & Disconnect	park	mntPARK=TRUE

Table 4: Hidden mntOp values sent from each mount form and corresponding Back-end driver commands

At the bottom of the page appears the date and time at which the mount status information was last updated. The purpose of this information is to allow the user to detect a problem. If the mount information has not been updated recently, then the user may suspect there is a problem with the back-end driver. This timestamp should be updated every time a command is sent from the web interface.

The mount Javascript also includes the function *validate_form* that is called when the *Slew To* form is submitted. This form simply checks the values of the position to ensure that the user did not enter non-numeric characters and that the values are within valid limits. A pop up window is displayed if the fields are not entered properly.

6.2.4. Observation Control Interface

The observation page allows the user control over each set of camera and optics. An observation is a specific period of time during which the cameras and optics will operate autonomously according to the user’s preferences.

The layout of the page resembles that of the mount page; the top section displays information about the current hardware status, and the bottom section provides settings that allow the user to control the hardware.

The whole page is organized such that all information and settings pertaining to the telescope camera and optics are in the left-most column; information and settings pertaining to the wide-angle camera are in the next column to the right. The right-most column of the page provides a preview of the last image acquired by each camera.

A screenshot of the observation control interface is provided in Figure 9.

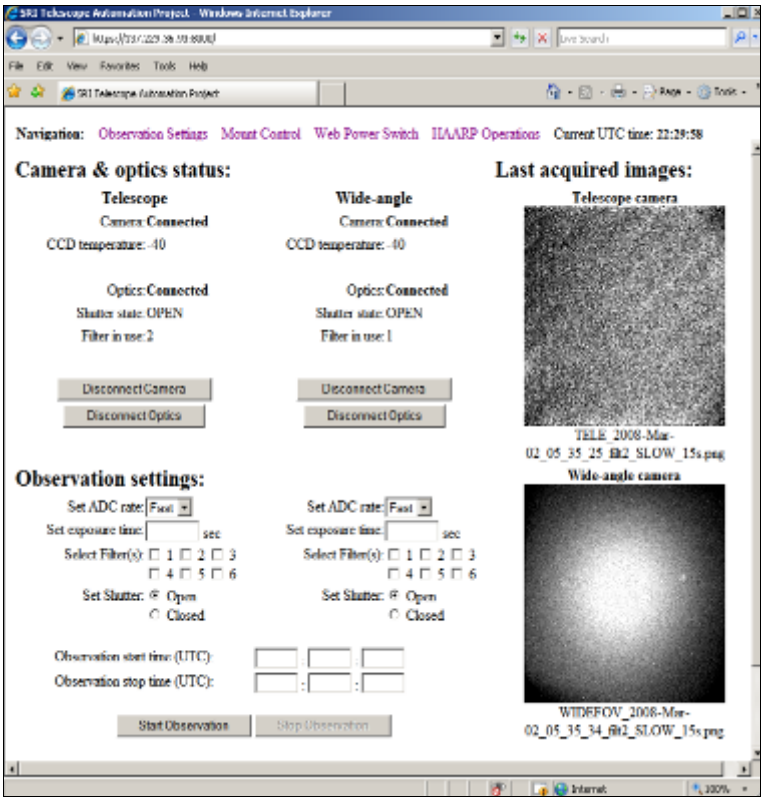


Figure 9: Observation control interface

The top portion of the left columns provides the status of each set of camera and optics. A piece of equipment can have a status of “Connected” or “Disconnected.” The status is displayed in bold text to stand out to the user as a disconnected piece of equipment would impede functionality. Below the status of each camera appears the CCD temperature, which is updated in near real time. Further down on the page appears information about each set of optics. The status display indicates whether each Smartmotor is connected, and below it appears whether each external shutter is open or closed and which filter is currently selected. If a piece of equipment is not connected, the status fields for that piece of equipment will indicate “n/a.”

Table 5 summarizes the div fields in the top portion of the observation page, as well as the XML parameters that they rely on.

Div ID	XML Tag	Div Content if Connected
teleCamStatus / waCamStatus	connectedState	“Connected” or “Disconnected” camera
teleCamCCDtemp / waCamCCDtemp	ccdTempState	Decimal value CCD temperature
teleOptShutter / waOptShutter	shutterState	“open” or “closed”
teleOptFilter / waOptFilter	filterState	integer value between 1-6

Table 5: Summary of div fields in observation page

Below the information displays are buttons that allow the user to connect or disconnect a piece of equipment; they change accordingly depending on the state of the equipment. If there is an observation in progress, and a piece of equipment is connected, then the *connect* and *disconnect* buttons for that equipment will not appear so that a user does not disconnect equipment during an observation. Each button is a separate form in the HTML file, and each button is placed in an appropriate div tag along with hidden fields that actually specify the command to the server. Table 6 specifies the values of hidden fields and the names of the div tag that display each button.

Button	Hidden field	Hidden field	Div tag
Telescope camera connect	equipment=teleCam	camConnect=TRUE | FALSE	teleCamConn
Telescope optics connect	equipment=teleOpt	optConnect=TRUE | FALSE	teleOptConn
Wide-angle camera connect	equipment=waCam	camConnect=TRUE | FALSE	waCamConn
Wide-angle optics connect	equipment=waOpt	optConnect=TRUE | FALSE	waOptConn

Table 6: Summary of hidden fields and div tags corresponding to each connect button

The next section of the page provides settings that allow a user to set up and control an observation. If the camera and optics are both connected for the telescope or wide-angle imager, then the user will see input fields for setting the ADC rate, exposure time, filter selection, and shutter state for each set of camera and optics. If either the camera or optics of an imager is not connected, then the input fields will be replaced by “n/a” for that imager only. It is possible to run an observation with only one set of camera and optics connected.

As the Javascript *observation.js* is running, it checks whether the status of a piece of equipment and the status of any ongoing observation has changed since the last time the XML file was updated. If so, then the function *controlField* is called to update the control panel. Otherwise, the control field is not updated every time the status XML is downloaded.

Beneath the imager-specific observation settings are fields for entering the start and stop time of the observation. This information is entered in the format *hh:mm:ss* in 24-hour coordinated universal time (UTC). If two cameras are used in an observation, then the start and stop times apply to both cameras. If a user does not specify a start time, then the observation starts immediately. If a stop time is not specified, then the observation continues until the “Stop Observation” button is pressed. This button is enabled whenever an observation is in progress.

As with the mount page, a function is called from the Javascript whenever the “Start Observation” button is pressed. This form only checks to make sure that a valid camera exposure was entered for each connected camera, as this value is necessary to operate the camera. Observation start and stop times are optional, so those input fields are not checked.

The user interface switches to the observation display whenever either of the imagers reports that it is conducting an observation. In this case, the user input fields show the settings that the user specified before starting the observation. These parameters were saved to the *obsUserPrefs.xml* file when the “Start Observation” button was pressed. If the imager is not connected, then the input fields for that imager will continue to display “n/a” while the other imager has an observation in progress. In this way, the content of the XML files dictates data and options that the user has available in various states of the system.

Table 7 summarizes the div fields that display the user input fields and XML fields that store user preferences for each field.

Div ID	XML Tag
tele|waObsADCrateDisp	camADCrate
tele|waObsExpTimeDisp	camExpTime
tele|waObsFilterDisp	optFilter
tele|waObsShutterDisp	optShutter

Table 7: Observation page status fields and corresponding XML tags

The right side of the observation page displays a PNG version of the last image that was acquired by each imager. These preview images are only 230x230 pixels in size, thus they are smaller than the full-size images. The process of encoding images in PNG is discussed in section 7.1.2. The filename of each image appears below it and specifies the date, time, filter, exposure time, and ADC rate when the image was taken, along with which camera took the image. Refreshing the images is handled by the observation page Javascript. While parsing the content of the status XML file, the Javascript compares the filename of the latest image with that of the previous one. If they do not match, it calls the function *updateImage* that instructs the browser to refresh the image with a given ID and download the image with the given filename. In the HTML file, the telescope image has the ID “teleCamLastImage,” and the wide-angle image has the ID “waCamLastImg”

Each of the preview images on the observation page is a link that the user can click. Clicking an image opens a new window that displays the latest acquired image for that imager in its full size of 512x512 pixels. These windows automatically refresh the image whenever filename of the last image changes in the corresponding status XML file. In this way, the user can view the images full-size as they are acquired by each camera.

The link to each full-size image is defined on the observation HTML page, and it is set up so as to instruct the browser to open a new window that is 530x575 pixels in size and does not include toolbars, which would unnecessarily crowd the user's desktop. Each image links to either *obsTelePic.html* or *obsWidePic.html*. These HTML files both import the Javascript called *imgBig.js* and begin by calling the function *imgProc*, which takes either "tele" or "wide" as an argument. The Javascript downloads the status XML file for that camera and checks whether the image filename has changed since the last time the XML file was downloaded. If so, the function tells the browser to download the image again, just like with the small preview images. At the end of the script, the *setTimeout* function calls the script again after five seconds. Both full-size image windows may be open at the same time even though they refer to the same Javascript, because the code will run concurrently for each window.

The user can, therefore, view data as it is acquired over the course of an observation and view the status of the hardware. The control interfaces provide the user with the ability to set preferences for running the hardware. The next section provides details about how the web interface operates and the configuration of the HTTP server.

6.2.5. Web Power Switch

As part of the telescope system automation, the cameras, optics, and mount were plugged into a web-controlled power switch. The *Web Power Switch* from Digital Loggers, Inc. is a power strip that includes a built-in HTTP server. A user can log into the device through a web browser and switch any of the eight switchable outlets on or off. The switch was configured to use a static IP address over nonstandard port 8001 that was opened to allow remote access from SRI. The web power switch can be used independently of the remote telescope system. Figure 10 shows the power switch after on-site installation.

Figure 10: Web Power Switch Installed On-Site

The web power switch includes a reset button that, when pressed, will reset the password and IP configurations to the default values. A crossover Ethernet cable may be used to connect the

web power switch directly to a computer in order to configure network settings. Once configured, the device can be connected directly to the network switch in the optics shelter with a straight Ethernet cable.

6.3. The HTTP Interface

The internet was the best choice of communication between SRI and HAARP as high speed connections are readily available at both facilities. Two-way communication between the hardware and user interface is achieved through HTTP. This format is recognized by all web browsers and allows text, images, and user input fields to be sent over the internet.

When a user points a web browser to a website, the browser is actually connecting to another computer at a specific IP address to ask for data to download over the HTTP protocol. Any computer that is connected to the internet can act as an HTTP server provided that the right software is running on it and listening for user requests. Once a user's browser asks for content, an HTML page, along with any Javascript or image files that it references, is sent from the server to the browser to be displayed for the user.

In the case of this project, we installed the Apache Server software on a computer inside the telescope dome. This way, a user at SRI can point a web browser to IP address of the computer, and the server will display the user interface in the browser. This section describes the configuration of the Apache server and details the script that it runs to process user input data.

6.3.1. Apache Server Configuration

The Remote Telescope System uses Apache Server version 2.2.6 to host the web-driven user interface. Apache is a popular open-source server program that is easy to set up and provides flexibility and security.

The Apache server was installed in the "Telescope Automation" directory on the wide-angle computer. All the files pertaining to the web-driven user interface were placed in the directory named *htdocs* located in the Apache directory. This directory is the root directory of the server, and any files placed in this directory can be accessed over the web. The Python CGI script, which is described in the next section, was placed in the *cgi-bin* directory under Apache.

The Apache server was installed as a service in the Windows XP computer, meaning that it will start automatically every time the computer is restarted. The server can be controlled through the Start menu, where an administrator will find buttons to Start, Stop, and Restart the server. The server must be restarted in order for any configuration changes to take effect.

Following the initial installation of the Apache server, we made several modifications to the configuration. All the modifications are made in the file named *httpd.conf*, which is located

inside the Apache subdirectory named *conf*. A copy of the actual Apache configuration file is provided in Appendix F. All modifications from this project have been commented with "te08."

The first modification was to change the socket port on which the server listens for incoming connections. The default HTTP port is 80, but we set Apache to listen on 7999 by altering the following line in the configuration file: Listen 7999. Using a non-standard port helps protect against unauthorized users gaining access to the system. In our case, port 7999 is actually blocked by HAARP's firewall, so no connections can be made to Apache; all incoming connections are handled by the secure virtual host, which is described below.

Another significant modification is the addition of password protection for the telescope control site. Adding user authentication requires the creation of an encrypted file that stores usernames and passwords, as well as a modification to the Apache configuration file. In order to generate a password file, open a Windows Command Prompt and navigate to the Apache *bin* subdirectory. There, enter the following command: `htpasswd –c <filename> <user name>`, where <filename> corresponds to the name of the password file, and <user name> corresponds to the username that must be entered to access the site. After entering the command, htpasswd prompts for the password that is to be created. Once it is entered, the password file appears in the *bin* directory. This file may be moved to another location, but for security it should not be placed in a web-accessible directory such as *htdocs*. Once the password file is in place, several lines must be added to the configuration file:

```
AuthType Basic

AuthName "SRI Access"
AuthUserFile "<path to password file>"
Require user <user name>
```

These lines should be placed in the directive called *Directory*, which provides the path to the *htdocs* root folder. Please see the Apache configuration file in the Appendix F for details.

The final major modification to our Apache setup is the addition of Secure Socket Layer, which enables communication over the HTTPS, or Secure HTTP, format that encrypts all communication between the browser and the server. Implementing this functionality required generating encryption keys, modifying the Apache configuration file, and modifying the Apache SSL configuration file.

In order to provide encryption, the server uses a public certificate and a private key. The certificate and key are generated together using a program called OpenSSL, which is open-source and available on the internet. To create the encryption keys, download and install

OpenSSL, and then navigate to the program's *bin* subdirectory in the Windows Command Prompt. Enter the following commands:

```
openssl req -new > example.csr -config openssl.cnf
```

```
openssl rsa -in privkey.pem -out example.key
```

```
openssl x509 -in example.csr -out example.cert -req -signkey example.key -
days 9999
```

Note that the certificate will be valid for the number of days specified in the *–days* argument in the last command. Once these commands have been executed, the key and certificate files will appear in the *bin* subdirectory. These files can be moved to any other directory, but should not be placed in a web-accessible directory for security. The *.rnd* file that is created with these encryption keys should be deleted for security.

The final custom modification to the Apache configuration is modifying the Apache configuration file and the SSL configuration file. The following two lines should be uncommented in the Apache configuration *httpd.conf*:

```
LoadModule ssl_module modules/mod_ssl.so
```

```
Include conf/extra/httpd-ssl.conf
```

Additionally, modifications must be made to the SSL configuration file, which is located in the *extras* directory in the *conf* subdirectory. This file is called *httpd-ssl.conf*, and the contents are included in Appendix F. The first modification is to change the port by setting the following directive: *Listen 8000*. Since port 8000 is open to computers with SRI IP addresses, all traffic will be immediately handled by the SSL virtual host, which means that it will be encrypted. The other two modifications are to specify the paths to the certificate and key files that were generated earlier.

These modifications will take place after the server has been restarted. At that point, configuration of the Apache server is complete, and the user can access the HTML files in the *htdocs* subdirectory and CGI scripts in the *cgi-bin* subdirectory. The next section explains the operation of the CGI scripts.

6.3.2. Python CGI Script

When a user submits any data by pressing a button on any submit form, the server extracts the values of the input fields and processes them as necessary. The processing of user input data is handled by a Common Gateway Interface (CGI) script. The name of the CGI script is *control.py*, and it is found in the *cgi-bin* subdirectory of Apache. The script was written in the Python language, and therefore, it is necessary that Python is installed on the server computer in order

for the script to function. We chose Python as the programming language for this script because it is a robust language with many included functions, such as extensive ability to parse strings and modules for TCP/IP communication. Additionally, a Python script does not need to be compiled before runtime, which makes it easier to modify and debug than some other languages. The entire code of *control.py* may be found in Appendix G**.**

Each form on an HTML page is declared with a form tag and contains the relative path to the CGI script as an attribute. The syntax is as follows:

```
<form action=”/cgi-bin/control.py”>
```

When the form is submitted, the server starts the script and passes the form data to it for processing. From there, the Python script performs several operations. The script retrieves the fields and input values from the form that the user submitted, translates field values into commands in the proper syntax for the back-end driver, transmits the commands over TCP/IP to the back-end driver, writes user preferences to an XML file, and finally instructs the user’s browser to return to the referring page so that an updated control interface will display once more.

The very start of the script sets the path to the Python executable on the system and imports the necessary libraries. The very next section provides paths to the XML files that store user preferences. These files are rewritten if the user starts an observation or commands the mount to slew. The IP address of each component and the address of the web-page are also specified in this section. The paths and IP addresses should be updated if the configuration of the system changes. No additional settings need to be changed after this point for the CGI script.

The first functional line of the script prints “'Content-type: text/html\n'.” This line tells the user’s browser that the server will be sending HTML data. Printing this line early on in the script allows subsequent messages that are printed by the script to be displayed by the browser.

When parsing the data submitted in the form field, the CGI script first looks for the “equipment” field. Since the names of the fields for each form are constant, as defined above, the script tries to acquire the value of each field that pertains to that piece of equipment. The hidden fields in the HTML files are actually in place to tell the CGI script which user input fields it needs to obtain for the given piece of equipment.

The field values that are read for each type of operation are accumulated into a single string of commands. As the value is read for a given field, a corresponding back-end driver command and the value are placed in the string. The commands are placed in the format “command1=value1\r\ncommand2=value2\r\n\xFF” where “\r” is the carriage return character

and “\n” is the new line character. The end-of-file character, “\xFF,” is appended to denote the end of the string, which is important when the string is transmitted to the back-end driver. In the case of observation start and stop times and mount positions, field values from three separate input fields get appended in the format “command=value1:value2:value3\r\n.” Filter selection boxes come into the script in the format [‘1’, ‘2’, ‘3’], but the tick marks and spaces are removed before sending the values to the back-end driver.

In the case where the “equipment” field value is “observation,” the script generates two separate scripts, one for the telescope equipment and one for the wide-angle equipment, as the two sets of equipment may have different user settings. Field names begin with “tele” or “wa” to specify which piece of equipment should receive the command. The field names that pertain to an observation begin with “obs,” and these commands are appended to both strings as observations occur concurrently with both cameras. The script looks to the hidden fields *teleSendTo* and *waSendTo*, which are “TRUE” or “FALSE” depending on whether the camera and optics are both connected, to determine whether to parse fields for that piece of equipment. If one of those fields is “FALSE,” then the user was not allowed to enter settings for it, and the script does not attempt to read the field values. The command *obsStart* is the last to be appended to the string as the back-end driver needs to have received all the other parameters before starting the observation. The commands are sent to the back-end driver in the order that they are entered into the command string.

The string can contain one or many commands, as long as they are in the right format. Connecting or disconnecting a piece of equipment sets up a string that contains only a single command, while starting an observation puts many commands in one string.

Once a string of commands has been generated, it can be sent to the back-end driver over the TCP/IP protocol. The script can tell which IP address to send to from the “equipment” and “sendTo” fields. The script sends to the appropriate back-end driver by calling the *sendString* function, which is defined at the top of the script. In the case of starting an observation with both cameras, the function is called once for one back-end driver and then for the other.

The *sendString* function uses Python’s built in TCP/IP socket handling procedures. The function takes the destination IP address and the string of commands as arguments. The function creates a network socket and attempts to use it to connect to the destination IP over port 5000. If a connection is acquired, then the function attempts to send the string. After sending the string, the function enters a loop in which it waits until it receives a null character across the connection. This loop ensures that the back-end driver receives all the commands before terminating the connection, which could some commands to be lost in the transmission. Finally, the function closes the socket and terminates.

After sending the strings over TCP/IP, the script continues by writing user preferences to an appropriate XML file. Settings are written to *obsUserSettings.xml* for observations and *mntUserPrefs.xml* for the mount. User settings are written only if the "equipment" field value was "observation" and the "obsStart" field was "TRUE," or if the "equipment" value was "mount" and the "mntOp" value was "slew." In the case of observations, the values for telescope and wide-angle fields are written only if the "sendTo" value for each piece of equipment is true; otherwise, a value of "none" is entered in that field. In the case of mount commands, the position values that the user entered are recorded.

The last section of the CGI script prints the following HTML code to the browser:

```
<HEAD>
<meta http-equiv="REFRESH" content="0;url=' + pageAddress + returnPage + '">
</HEAD>
```

These lines, which make up the header of the HTML file that is being sent to the browser, instruct it to refresh to the given URL immediately. The script inserts the address of the web server and which control page the user should be returned to. Then, by the time the browser downloads the new page, the latest data has already been written to the XML files and the user sees that the interface responded to the last set of commands. In the interim between executing the script and before the new page is downloaded, the user will see the word "Processing…" The purpose of this message is merely to provide some brief content until the page reloads, and this message may be changed by modifying the Python code.

The *control.py* CGI script also takes advantage of Python's built-in error handling. At several points in the script where an operation might fail, the commands *try* and *except* specify which commands Python should attempt and what to do in case they fail. Specifically, this error handling occurs when the script attempts to connect and send a string to the back-end driver over TCP/IP and when the script attempts to write user settings to the XML file. In the event that the file cannot be written, or a back-end driver cannot be reached, the user will see a helpful error message rather than the default "Internal Server Error" message that is given by Apache.

Whenever code inside the *try* command fails, Python immediately jumps to the *except* command and executes the code it contains to handle the error. In the case of this CGI script, each *except* command generates a message about which part of the script failed and then calls the *errorPrint* function that is defined near the beginning of the file. This function takes the error message, which was stored in a string called *error*, and prints it to the screen before closing the script. The CGI script terminates after printing the error message.

In this way, the data a user specifies through the form-based web interface gets received by the Apache server, parsed from the form fields, transmitted to the back-end driver, and written to the XML file before the user's browser returns to the interface page. The next section discusses how the back-end driver uses this data to control the hardware.

6.4. Back-end Driver

As mentioned previously, the back-end driver is the heart of the system. It accepts user commands relayed from the website by the Apache server or sent directly via a telnet, processes them, and then passes them on to their corresponding sub-modules. The commands range from changing various hardware settings to signaling the beginning of an autonomous observation. The back-end driver, as well as all other software modules used in conjunction with the back-end driver, was written in C++ using Microsoft Visual Studio 2005 on a computer running Windows XP Professional.

The back-end driver is a multi-threaded program consisting of one thread for accepting TCP/IP requests, one for running observations, and one for controlling the mount. The observation thread communicates with the hardware to select the appropriate filter, acquire an image, and then write the image to both supported formats. It continues to repeat the process until either the specified end time is reached or a stop is requested by the user. The mount operation thread starts an obersvation by calling external Visual Basic scripts. Threading is required here because these scripts do not return control back to thebackend driver until they complete moving the mount. In all cases, threads are created as needed and then closed.

The back-end driver was divided into much smaller, task-specific modules to ease implementation. These modules are explained in the following subsections.

6.4.1. TCP/IP Server Module

The TCP/IP Server module provides a simplified interface to the Windows TCP/IP API. After instantiating the module, the user makes a function call to *initServerAndListen()*, shown below:

```
int initServerAndListen();
/**
*   Creates a socket, binds it to the port specified when instantiating
* this class, and begins listening for a client connection request.
*
* INPUT:
*   none
* RETURNS
*   0 in success, error code on failure.
*/
```

The function takes all the necessary steps in setting up Windows socket for TCP/IP server communication. This includes initializing the WinSock library, creating the socket, binding it to

the port specified when instantiating the module, and instructing the socket to listen on the given port for client connection requests. After a successful initialization, the user accepts client requests using *acceptClient()*, shown below:

```
int acceptClient();
/**
*   Checks for client awaiting connections.  If client request is found,
* that client is accepted and a socket is created for communication with
* client.  This function continues to wait (blocks) if no request is found.
*
* INPUT:
*   none
* OUTPUT:
*   0 on success, error code on failure
*/
```

After accepting a client, the user calls the *recvFromClient()* and *sendToClient()* functions to receive and send data from and to the client, respectively:

```
int recvFromClient(char *buf, int bufLen, char* delim);
/**
*   Receives data sent from accepted client.  This function polls the
* TCP/IP buffer until either one of the specified delimeters is reached
* or the buffer being written into is full.
*
* INPUT:
*   buf - pointer to buffer which received data is written to
*   bufLen - length of buffer being written to
*   delim - cstring containing all delimiters  ex: ";\r\n"
* RETURNS:
*   number of bytes read (>= 0), or error code (< 0)
*/

int sendToClient(char *buf, int bufLen);
/**
*   Sends data to accepted client.  Specified number of bytes in
* specified buffer are transmitted.
*
* INPUT:
*   buf - pointer to buffer containing data being transmitter
*   bufLen - number of bytes to transmit
* RETURNS:
*   0 on success, error code on failure
*/
```

The *recvFromClient()* will read characters from the TCP/IP input buffer until it receives *bufLen* bytes, an error is encountered (such as a closed connection or “end of file” character - EOF), or until it reads in the specified delimiter. The *sendToClient()* will send *bufLen* characters of the string *buf*. Once the user is done with the communication link, they use the *closeClientConn()* function to terminate the link:

```
  int closeClientConn();
  /**
  *   Closes connection with accepted client.  Because only one client
  * is supported by this class (one at a time), this function should be
called
  * right before attempting to listen for another client request.
  *
  * INPUT:
  *   none
  * OUTPUT:
  *   always 0
  */
```

6.4.2. XML Module

The XML module handles all XML file writing and is implemented in a tree format for expandability. A data structure containing the current value of each XML tag value is created, as well as a node for each tag in the XML file. Each node has a tag name and either a tag value or a list of sub-nodes (children). The tag name is assigned to the node at creation. However, because the tag value may change frequently, each node is assigned a reference to an entry in the data structure containing all the settings. The creation of the tree is illustrated in the example shown below:

```
<name1>
      <name2>value2</name2>
      <name3>value3</value3>
</name1>
```

The above XML snippet contains three tags: name1, name2 and name3. The tags name2 and name3 are children of the name1 tag. Because name1 is a parent node, it has no value and instead has children. Similarly, because name2 and name3 are not parent nodes, they have values and no children. To create a tree with the properties shown, we use first create the nodes, then use the *addChild()*:

```
  int addChild( XmlNode* newChildNode );
  /**
  *   Consumes a pointer to an XmlNode and makes that node a child of the
  * current node.
  *
  * INPUT:
  *   newChildNode - new node to be added as a child
  * RETURNS:
  *   0 on success, 1 on failure
  */
```

Once the tree has been implemented, to write the XML tree to a file the user would simply call the *printTree()* function:

```
int printTree( const char* filename );
/**
*   Iterates through XML tree with head node this and prints the tree
* to a file with the name filename.
*
* INPUT:
*   filename - name of the file to be written
* OUTPUT:
*   0 on success, 1 on failure
*/
```

This function iterates through the tree and prints the XML tag names and values in the correct order with the proper indentation.

6.4.3. Observation Module

The observation module is the interface between the back-end driver and the hardware interface modules. As commands are received by the back-end driver, they are passed to the observation module. The entry point to the module is the *processCommand()* function:

```
char* processCommand( char* cmd );

/**
*   Consumes a command and redirects it to the correct function.  If the
* command is valid, a return message is relayed from the called sub-function
* to the caller of this function.  If the command is not valid, an 'invalid
* command' string is returned.  The commands received are not case-sensitive.
*
* INPUT
*   cmd - The command to be processed.  Must be a cstring in the following
*         format:  "param=value\r....\0".  Everything between the first '\r'
*         and the first '\0' is ignored.
* RETURNS
*     Cstring containing information about the status of the processed command,
*   or an invalid command string if unable to process given command.
*/
```

This function first converts the entre command to upper-case to remove any case-sensitivity. Next, the command is checked for the correct format: *parameter=value'\r'*. If the command passes both tests, the *parameter* is located in a lookup table containing a corresponding function which carries out the command. If the *parameter* is found in the lookup table, then the function associated with that parameter is called and passed the *value*. Valid commands, values, and their significance are listed in Table 8.

Parameter	*Value*	*Action performed*
OPTCONNECT	“TRUE”	Connect to optics
	“FALSE”	Disconnect from optics
OPTSHUTTER	“OPEN”	Open the shutter
	“CLOSE”	Close the shutter
OPTFILTER	“N”	Set filter to N (digit)
	“[a,b,c]”	Set filter sequence to a, b, then c
CAMCONNECT	“TRUE”	Connects the camera
	“FALSE”	Disconnects the camera
CAMEXPTIME	“N”	Sets exp time to N seconds
	-	-
CAMADCRATE	“SLOW”	Sets cam ADC rate to slow
	“FAST”	Sets cam ADC rate to fast
OBSSTARTTIME	“HH:MM:SS”	Sets obs start time to HH:MM:SS
	-	-
OBSSTOPTIME	“HH:MM:SS”	Sets obs stop time to HH:MM:SS
	-	-
OBSSTART	“TRUE”	Starts an observation
	“FALSE”	Stops an observation
MNTCONNECT	“TRUE”	Connects the mount
	“FALSE”	Disconnects the mount
MNTHOME	“TRUE”	Homes the mount
	-	-
MNTAZI	“DEG:MIN:SEC”	Sets azimuth to DEG:MIN:SEC
	-	-
MNTALT	“DEG:MIN:SEC”	Sets altitude to DEG:MIN:SEC
	-	-
MNTSLEWTO	“TRUE”	Slews the mount
	-	-
MNTABORT	“TRUE”	Aborts the current slew
	-	-

Table 8 Back-end Driver Commands

The value is verified in each sub-module to ensure that an invalid entry was not made. The verification is explained in the comments of the actual code (Appendix H).

6.5 Hardware Interface Modules

The back-end driver’s Observation Class can interact with hardware components of the system through high-level function libraries called Hardware Interface Modules. Each of these modules contains a library of lower-level code that interacts with the manufacturer’s software interface for each device. . All the details of communicating with the hardware using the various

interfaces are hidden to the back-end driver by wrapping them up in hardware interface modules, explained in the following sections.

6.5.1 Optics

The optics control module provides the user with a simplified interface to the RS232 communication used to change and query the status of the hardware. First the user must open a serial communication link with the optics using *open()*:

```
/* open()
 *
 *    Opens and configures a connection with specified com port.  If an
 *  incorrect com port is specified (not an integer > 0), COM1 is used
 *  by default.  ** CURRENTLY HARDCODED TO ONLY ACCEPT COM 1-4 **  If
 *  anything above COM4 is requested, COM1 is used by default.
 *
 * Input:
 *  portNum - integer specifying which com port to use (COM X)
 * Returns:
 *  0 on success, 1 on error
 */
int open(int portNum);
```

After opening the connection, the user may begin sending commands to the SMARTMOTOR system controlling the optics. This is done through two get/set pairs of functions; one for the shutter *getShutter()/setShutter()* and one for the filter wheel *getPosition()/setPosition()*:

```
/* getPosition()

 *
 *    Sends command to ask the SMARTMOTOR system for its current filter
 *  wheel position and then waits for a reply.  Information is extracted
 *  from reply and current position is returned to calling function.
 *
 * Input:
 *  none
 * Returns:
 *  integer representing current filter position (1 -> n)
 */
int getPosition();

/* setPosition()
 *
 *  Sends commands to move filter wheel to specified position.  Invalid
 * position values are not checked here as they are handled by the
 * SMARTMOTOR system itself.  Waits for the SMARTMOTOR system to send
 * its confimation.  To confirm positioning getPosition() should be used.
 *
 * Input:
 *  pos - integer specifying which position to move to.
 * Returns:
 *  new position of filter wheel
 */
int setPosition(int pos);

/* setShutter()
```

```
 *
 *    Sends commands to change the shutter to specified state.  If the
 *  specified state is anything but open (including invalid values), a closed
 *  state is assumed (for safety).
 *
 * Input:
 *  open - integer specifying which position to place shutter in. 1 for opened,
 *         anything else for closed.
 * Returns:
 *  integer representing current shutter status ( 1 = open, 0 = closed )
 */
int setShutter(int open);

/* getShutter()
 *
 * S  ends commands to ask the SMARTMOTOR system for its current shutter
 * position and then waits for a reply.  Information is extracted from
 * reply and current shutter position is returned to calling function.
 *
 * Input:
 *  none
 * Returns:
 *  integer representing current shutter status ( 1 = open, 0 = closed )
 */
int getShutter();
```

The optics control module provides the user with a simplified interface to the RS232 communication used to change and query the status of the hardware. First the user must open a serial communication link with the optics using *open()*:

```
/* open()
 *
 *    Opens and configures a connection with specified com port.  If an
 *  incorrect com port is specified (not an integer > 0), COM1 is used
 *  by default.  ** CURRENTLY HARDCODED TO ONLY ACCEPT COM 1-4 **  If
 *  anything above COM4 is requested, COM1 is used by default.
 *
 * Input:
 *  portNum - integer specifying which com port to use (COM X)
 * Returns:
 *  0 on success, 1 on error
 */
int open(int portNum);
```

After opening the connection, the user may begin sending commands to the SMARTMOTOR system controlling the optics. This is done through two get/set pairs of functions; one for the shutter *getShutter()/setShutter()* and one for the filter wheel *getPosition()/setPosition()*:

```
/* getPosition()

 *
 *    Sends command to ask the SMARTMOTOR system for its current filter
 *  wheel position and then waits for a reply.  Information is extracted
 *  from reply and current position is returned to calling function.
 *
 * Input:
 *  none
```

```
 * Returns:
 *  integer representing current filter position (1 -> n)
 */
int getPosition();

/* setPosition()
 *
 *  Sends commands to move filter wheel to specified position.  Invalid
 * position values are not checked here as they are handled by the
 * SMARTMOTOR system itself.  Waits for the SMARTMOTOR system to send
 * its confimation.  To confirm positioning getPosition() should be used.
 *
 * Input:
 *  pos - integer specifying which position to move to.
 * Returns:
 *  new position of filter wheel
 */
int setPosition(int pos);

/* setShutter()
 *
 *    Sends commands to change the shutter to specified state.  If the
 *  specified state is anything but open (including invalid values), a closed
 *  state is assumed (for safety).
 *
 * Input:
 *  open - integer specifying which position to place shutter in. 1 for opened,
 *         anything else for closed.
 * Returns:
 *  integer representing current shutter status ( 1 = open, 0 = closed )
 */
int setShutter(int open);

/* getShutter()
 *
 * S  ends commands to ask the SMARTMOTOR system for its current shutter
 * position and then waits for a reply.  Information is extracted from
 * reply and current shutter position is returned to calling function.
 *
 * Input:
 *  none
 * Returns:
 *  integer representing current shutter status ( 1 = open, 0 = closed )
 */
int getShutter();
```

6.5.2 Cameras

The list summarizes the camera functionalities that were implemented to form the camera module. The source code for all these functions is available in Appendix H.

connectCam(): This function initializes the PVCAM library and connects to the first camera in the list.

disconnectCam(): This function un-initializes the PVCAM library and disconnects the connected camera.

readTemp(): This function returns the current temperature (16 bit value) of the camera controller.

setTemp(int): This function sets the camera controller temperature to the user specified value.

setExposureTime(int): This function sets the exposure time for the camera to the user specified value. The exposure time should be specified in seconds.

setADC(int): This function sets the ADC rate of the camera to the user specified value. There are only two possible values, fast or slow. A fast rate corresponds to 10 MHz while a slow rate corresponds to 100 kHz.

readShutter(): This function reads in the camera's shutter state. There are three possible states: Normal, Disabled Open and Disabled Closed. A Normal state corresponds to the shutter opening before an exposure and closing right after. A Disabled Open mode corresponds to an open shutter state at all times whereas a Disabled Closed mode corresponds to a closed shutter at all times.

setShutter(int): This function sets the shutter state to one of the three possible modes mentioned above.

takeImage(): This function allows the user to take pictures with the camera. After having configured all the settings, this function will capture an image, put the raw data in a file and then append the WinView header so that the pictures may be viewed using the WinView software.

6.5.3 Mount

The back-end driver controls the mount by sending commands to TheSky6, the software that normally is used to control the mount. Since the control format for the mount is proprietary, we were not able to create a custom interface module. However, TheSky6 can operate in a scripted mode where its functions are controlled by a Visual Basic script (VBscript). A VBscript is based on the Visual Basic programming language, but the source code does not compile before runtime. No special software is needed to run a VBscript, as the execution of the script is handled by the Windows Scripting Host (WSH).

TheSky6 includes a number of object classes that allow other programs to communicate directly with the software through the use of VBscripts. Among these objects is the *RASCOMtele* object, which is used extensively in the mount interface module. This object allows scripted operation of any mount that is compatible with TheSky6. The mount module uses a subset of the mount

commands that are available through this object. When one of these scripts is called, TheSky6 automatically starts running on the host PC.

Each of the VBscripts that makes up the mount module begins by instantiating the *RASCOMtele* object with the following line:

```
Set objTele = WScript.CreateObject("TheSky6.RASCOMTele")
```

Once the object has been instantiated, any commands available in the object may be called in the format: *objTele.command().* At the end of the script, memory that was allocated to the object is cleared with the command: `objTele = Nothing`

When calling one of the VBScripts to perform a mount function, it should be noted that the Windows Scripting Host defaults to running VBscript files in the Windows GUI. Any messages that return from the script are thus displayed in pop-up windows; this behavior is inconvenient for a remote interface. In order to run the script from the console and have standard output messages return to the console window, each script must be called with the following convention: *cscript //nologo command.vbs*. In fact, the mount module executes each script by including this format in the *system()* command.

Several of the scripts may be called while another operation is in progress. These scripts are *abort.vbs, getposition.vbs,* and *isconnected.vbs*. Since some operations take several seconds, like slewing or homing, it is beneficial to update the status of the device while it is performing the operation. In this case, the second command may be called from a second console window before the first returns. Or, in the case of a C++ program, the second command may be called from a second thread running in parallel with one that called the first command.

The source code for each of the mount VBscripts may be found in Appendix I. The following is a listing of all the VBscripts that make up the mount class:

abort.vbs – When this script is called, the mount halts any ongoing operation immediately. This command can be called while the mount is slewing, homing, or parking.

Calls:	*objTele.abort()*
Returns:	"0" if the script executed properly
	"Mount Error!" if there was a problem executing the script

connect.vbs - Script starts TheSky6, which proceeds to establish communication with the mount. Once these commands have been issued, the script checks whether a connection was in fact established or whether an error was encountered. If the connection was successful, then the script tells the mount to operate in asynchronous mode, which allows it to respond to inquiries, such as obtaining current position, even if a slew is in progress.

Calls: *objTele.Connect()*
objTele.SetTracking(0, 1, 0, 0)
objTele.Asynchronous = 1
Returns: "0" if the script executed properly
"Mount Error!" if there was a problem executing the script

getposition.vbs - Script returns the current position of the mount in terms of azimuth and altitude. Azimuth and altitude return as degrees, minutes, and seconds with each value rounded to the nearest integer. Note that the RASCOMtele commands return degrees with a decimal, which this script converts to minutes and seconds. This script can be called any time that the mount is connected, even if another operation is ongoing.

Calls: *objTele.getAzAlt()*
Returns: "aziDeg=359;59;59;aziAlt=90;0;0;(null)(cr)" if executed properly
"Mount Error!" if there was a problem executing the script

home.vbs – Script moves mount to the home position.

Calls: *objTele.FindHome()*
Returns: "0" if the script executed properly
"Mount Error!" if there was a problem executing the script

isconnected.vbs – Script checks whether TheSky6 is communicating with the mount and returns the result.

Calls: *objTele.IsConnected*
Returns: "1" if the mount is connected
"0" if the mount is not connected
"Mount Error!" if there was a problem executing the script

park.vbs – Script moves the mount to the preset park position and disconnects it from TheSky6.

Calls: *objTele.Park()*
Returns: "0" if the script executed properly
"Mount Error!" if there was a problem executing the script

slewto.vbs – Script slews the mount to a given azimuth and altitude. This script must be executed with six arguments that define the azimuth and altitude in degrees, minutes, and seconds. Note that the *RASCOMtele* command called by this function takes in Azimuth and Altitude in decimal degrees, so this script converts the user input from degrees, minutes, and seconds into the proper format. The arguments must be separated by a space and provided in the following order:

Calls: *objTele.SlewToAzAlt(Azimuth, Altitude, " ")*
Arguments: aziDeg aziMin aziSec altDeg altMin altSec

Returns: “0” if executed properly
“Syntax Error!” if the arguments are out of range
“Mount Error!” if there was a problem slewing the mount

7. Image Encoding and File Transfer

The back-end driver processes the raw data acquired from the camera to produce two separate files. WinView format files, with a .SPE extension, include all the raw data bytes and a header that allows the file to be opened by the WinView image analysis software. Data stored in this format is sent back to SRI for analysis. The back-end driver also creates a preview of each image that is encoded in a web-friendly image format. These images are displayed in the user interface and made available to the HAARP website server for public display.

This section explains the processing of image data from the time it is read off the camera's CCD, until it arrives at the final destination for each image format.

7.1 Format Conversion

The back-end driver receives data from the camera as a vector of unsigned 16-bit integer values, each of which corresponds to the intensity of a single pixel. Each 16-bit value can represent decimal numbers from 0 to 65535, which corresponds to black and white, respectively; each value in between represents a unique shade of gray. As the size of each camera's CCD is 512x512 pixels, there is a total of 262,144 pixel values or 524,288 bytes of data per image. The next two subsections explain how this raw data is converted to usable WinView and PNG image formats.

7.1.1 WinView Format for Data Analysis

The raw data containing the pixel values coming from the camera is first written to a binary file by the *takeImage* function in the *CamCtrl* module. In order for this data show up as a proper image in WinView, the .SPE format has to be supported. This is achieved by appending an appropriate WinView header in front of the pixel data. The WinView header format is available in Appendix J. The .SPE format is arranged such that the first 4100 bytes of data denote the header for the image. Any following data are the raw pixel values collected directly from the camera.

Having looked at the 4100 bytes of data in the header closely, we determined that there were only a handful of parameters that needed to be changed in order to adapt the header to our data. In order to display the raw data in the .SPE format, we came up with a "generic" WinView header for all our data, which is generated by the back-end driver for each image. We were able to do this by taking a regular WinView image (having the .SPE extension) and stripping its header from the file. Once we had its header bytes written to a binary file, we were able to manipulate the necessary parameters of the header data to conform to the raw data taken by our cameras. The Matlab code that generated the generic header file is attached in Appendix K. The parameters that were adjusted for this project are listed as follows:

X dimension: This is the actual number of pixels on the x-axis. This parameter is located at byte offset 42. The data type for this is a *word* which means that it is two bytes long. Thus, we had to set bytes 42 and 43 of the header manually. For our purposes, we set the x dimension to 512 since the cameras are a (512 x 512) 16-bit value.

Y dimension: This is the actual number of pixels on the y-axis. This parameter is located at byte offset 656. The data type for this is a *word* which means that it is two bytes long. Thus, we had to set bytes 656 and 657 of the header manually. For our purposes, we set the y dimension to 512, just as for the x dimension.

Experiment format: This parameter specifies the data type for the pixel values. This is essentially the number of bytes per pixel for the camera. The parameter is located at byte offset 108 and is of type short, which is two bytes long. Thus, we had to manually set bytes 108 and 109 of the header. As per the header information, a value of 0 denotes a *float* value with 4 bytes per pixel, a value of 1 denotes a *long* value with 4 bytes per pixel, a value of 2 denotes a *short* value with 2 bytes per pixel and a value of 3 denotes a *unsigned short* with 2 bytes per pixel. Since the cameras used in this project use unsigned shorts, the parameter value is 3.

Number of frames: This parameter holds the number of frames contained in an image taken by the camera. It is located at byte offset 1446 and is of type *long* (4 bytes). Thus, bytes 1446, 1447, 1448 and 1449 had to be manually set to configure this parameter. For our purposes, this value is always one since the back-end driver writes each image to its own WinView file. However, the software may be reconfigured to the place multiple images in a single file.

Figure 11 shows screen captures of WinView display of the images in WinView format:

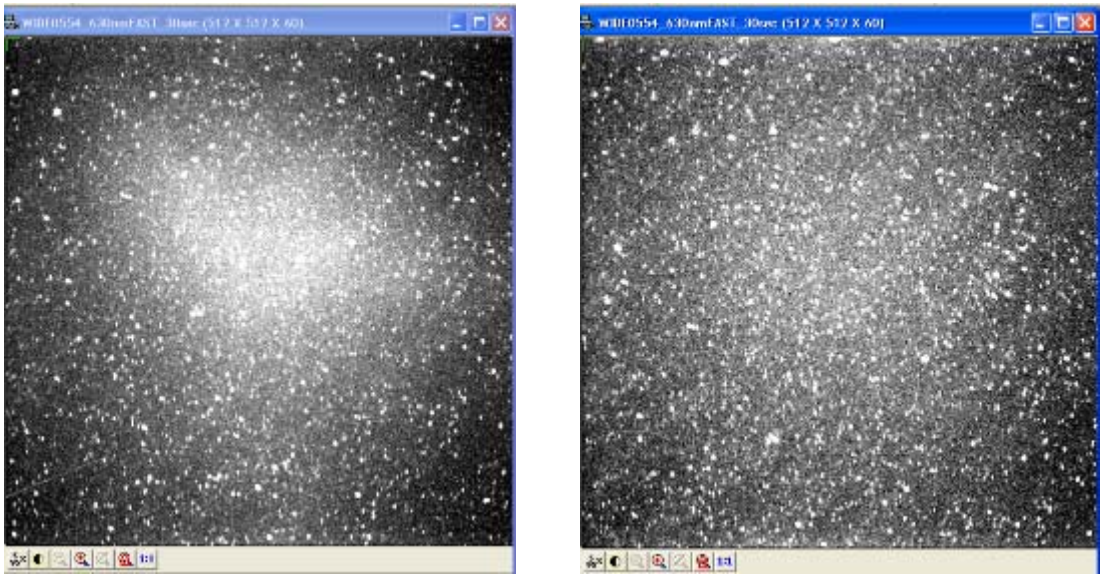


Figure 11: 16-bit WinView images with "5-95" applied. With airglow (left), and without airglow (right).

7.1.2 PNG Format for the Web

As image data is read by the computer off the camera's CCD, it is stored as a vector of bytes. Every two bytes correspond to a pixel intensity, which is displayed as a shade of gray on the computer screen. However, this format cannot be interpreted by most web browsers. In order for the images to be visible through a user-friendly interface such as the internet, they must be encoded in a web-friendly format. In the telescope automation system, each image is translated into the Portable Network Graphics (PNG) format. PNG is an open-source image format that offers lossless compression and is compatible with most internet browsers.

Converting the HAARP images to the PNG format presented two separate challenges. The first problem was encoding the data in the complex format, which the project team resolved through the use of a free library called LodePNG. The second challenge was processing the image so that the extremely faint airglow would be visible.

While the back-end driver obtains 16-bit data from the camera, these values are scaled down to 8-bits before conversion to PNG. In 8-bit format, each pixel can have an intensity from 0 to 255, which corresponds to a gray color between black and white. The team discovered experimentally that this conversion helps reduce graininess of the images. The project lead at SRI agreed that since the PNG images are provided only as a web-friendly preview of the data, as opposed to the WinView images that are used for analysis, the reduction in intensity resolution is acceptable in these images.

The LodePNG library includes functions that encode and decode data to and from the PNG format and converts image files between the various color types supported by PNG. The program was written by Lode Vandevenne and is available online; we incorporated this code into our project to handle PNG conversion. The library download includes three C++ source files called *lodepng.cpp*, *lodepng.h*, and *lodepng_examples.cpp*. Complete documentation of the program can be found in the header file, which can be found in Appendix L.

In the back-end driver, the LodePNG library is employed by the function *writeImageToPNG*, which is part of the *Observation* class of the back-end driver. The function sets the LodePNG encoder to generate a compressed 8-bit grayscale image using the following lines:

```
LodePNG::Encoder encoder;               // create encoder

encoder.infoRaw.color.colorType = 0;    // set raw data color to grayscale
encoder.infoRaw.color.bitDepth  = 8;    // set raw data bit-depth to 8-bit
encoder.infoPng.color.colorType = 0;    // set output color to grayscale
encoder.infoPng.color.bitDepth  = 8;    // set output bit-depth to 8-bit

encoder.getSettings().zlibsettings.useLZ77 = true;  // enable compression
encoder.getSettings().zlibsettings.windowSize = 32768; // default
encoder.getSettings().zlibsettings.btype = 0;       // compression settings
```

The writeImageToPNG function reads the low-byte of each value of the vector of 16-bit values one-by-one into a second vector, which contains the values for processing. The raw data values are in little-endian encoding, meaning that the least significant byte has the lower address. Through trial and error, we discovered that the re-encoded images look best when the 8-bit pixel value is immediately set to 255 if the high-byte of that pixel has a value greater than x02 (hexadecimal). Otherwise, the value of the low byte for the pixel is copied directly into the 8-bit vector. This operation represents the conversion from 16-bit representation to 8-bit representation. The rest of the function performs image processing to enhance the visibility of airglow.

Airglow is very faint; even in images acquired with a long exposure, the airglow appears at intensities slightly higher than the noise from the CCD. Additionally, only a small portion of the range of intensities that an 8-bit value can encode is used to show the airglow. Aside from the black sky and faint airglow, the rest of the image is composed of bright stars and CCD noise. If an unprocessed image is viewed, only black and a few stars that appear as small white specs are visible. To make the airglow visible, the range of intensities that represent it must be expanded to include more of the shades of gray that are available in an 8-bit image. WinView has a built-in function called “5/95” that accomplishes this processing, so it is not necessary for the WinView data files; the PNG images require that a similar operation be performed to make airglow appear.

The *writeImageToPNG* function performs an algorithm on each image to adjust pixel intensities so as to make airglow most visible. The algorithm determines how many pixels there are at each intensity, changes values of the dimmest 5% of pixels to 0 (black) and the brightest 5% of pixels to 255 (white), and scales the intensities of the remaining pixels linearly across the full range of values.

The first step to redistributing pixel intensities is to create a histogram of the 8-bit image data, which tells how many pixels there are for each intensity. The function has an array of 256 integers called *histogram* that is populated by using the value of each pixel as the index of the array, and incrementing the array index by 1 for each pixel, as follows:

```
for( i = 0; i < PNG_W * PNG_H; i++ )
   histogram[cImage[i]]++;
```

Once a histogram has been calculated, it is possible to calculate the 5% dimmest and brightest pixel values in the image, which are referred to as *lowP* and *highP* in the program, respectively. The pixels with 5% of the lowest nonzero values in the image are likely to contain just noise, and not actual airglow. The pixels with 5% of the highest intensities are likely to be bright stars, so it is beneficial to change their values to the maximum and use that extra bit of dynamic

range to have more intensities available for displaying airglow. The value of lowP is calculated by performing a cumulative sum on each position of the histogram array, starting with 1 so as to not include pixels of value zero in the calculation. The index of the histogram array at the point where the cumulative sum is just above 5% of the total number of pixels becomes the *lowP*. A similar operation is used to calculate *highP*, except that the algorithm starts at the highest intensity and counts down until 5% of the total number of pixels has been reached.

Once the cutoff values have been calculated, the algorithm then proceeds by changing the value of each pixel, one by one. Pixels with values above and below the cutoffs are adjusted to 0 or 255, and the other ones are scaled according to Equation 1:

$$y = \frac{(x - lowP) \times 255}{(highP - lowP)}$$

Equation 1: Linear redistribution of pixel values within 5%-95% range

The equation produces a pixel intensity 0 < y < 255, which has a value that is proportional to the value of the input intensity *lowP* < x < *highP*. In this way, pixel values are adjusted across the full range of 8-bit values to show airglow as clearly as possible.

Finally, the *writeImageToPNG* function sends the vector of processed 8-bit values to the LodePNG encoder to generate a PNG image file. Figure 12 shows the WinView images from above re-encoded into PNG with pixel value scaling applied. One image has airglow, and the other does not; but before processing, both images were completely black.

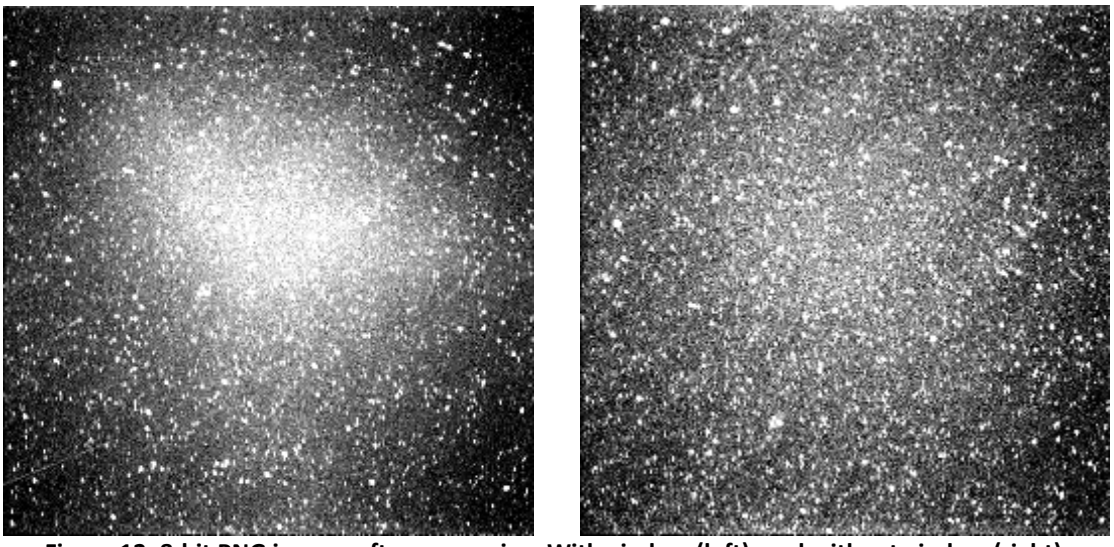


Figure 12: 8-bit PNG images after processing. With airglow (left), and without airglow (right).

The *writeImageToPNG* function processes the raw data from the camera and generates images that appear very similar to the raw WinView format images. These PNG images display airglow clearly and can be incorporated into a webpage for display.

7.2 File Transfer Automation

One of the project requirements was to transfer the images captured by the cameras at HAARP onto a local computer at SRI. Andrew Young of SRI suggested the use of the free DeltaCopy program which he was already using for a different project.

The DeltaCopy program allows for a relatively fast method of backing up data over Windows NT platforms. It is fast because it backs up data incrementally, which means that it compares the source and destination files (inside a directory) and only copies over any discrepancies (bytes that may have been modified or added since the last backup session). Thus, it saves some time since it does not copy the entire file after only a few modifications. The program is based on the *rsync* program for Unix/Linux/BSD systems.

Even though DeltaCopy essentially follows a Client/Server model on a Windows platform, we had to make some modifications on the server side since the images were being uploaded onto a Linux machine in California. This modification involved tweaking the *rsyncd.conf* file on the Linux machine for the cross-platform upload to run smoothly. The following directives were changed in this configuration file:

```
path = /tmp/test
read only = false
```

The DeltaCopy Client program was installed on the computer at HAARP to upload the images onto a Linux server. The DeltaCopy Client program on the HAARP computer has been scheduled to upload any new images every 5 minutes from 2am (UTC) to 9am (UTC) everyday. The folders that are transferred to the SRI server are the *WinView* image folders for both back-end drivers.

7.3 Web Posting

One of the desired outcomes of the project was the ability to post images from the telescope system to the internet for public display. Since the observation page has to display images from the telescope in a web browser, the images are already encoded in a format that can be available for display on any other website. The PNG versions of each image are stored in a network shared folder that has been made available to the HAARP servers. The names of the network folder for each camera are as follows:

```
\\WideFOV\telescope_web_images\
\\WideFOV\wideangle_web_images\
```

These network folders can be accessed internally from the SRI network. The project team spoke with Mike McCarrick, a manager at HAARP, about configuring the public web server to include these images. The public display of images is still pending at the conclusion of the project, but once HAARP is ready to do so, the images will already be in a web-friendly format.

8. Potential Solutions for Dome Automation

One part of the desired functionality of an automated telescope system was the ability to control remotely the dome that houses the telescope system components. While researchers at SRI now are able to control the mount, cameras, and optics remotely, they still are not able to gather any real data unless someone on-site opens and positions the dome slit. Additionally, someone on-site must ensure that the dome is closed at the end of the observation, or in the event of precipitation, in order to protect the imaging system.

Incorporating the ability to control the dome remotely to the remote telescope control system offers several benefits. The operators of the telescope system pay close attention to the conditions as they perform an observation. These operators would notice threatening weather conditions from the images, even if operating the system remotely. An automated dome would provide these users with the ability to shut the slit immediately. Otherwise, the telescope would be exposed until someone is reached on-site to close the dome manually. Additionally, the researchers that use the dome can shut it immediately when their work is complete, thereby reducing the likelihood that someone on-site will forget to shut it overnight. Operation of the dome can be tied to the automated imaging provided by the back-end driver, in which case the conclusion of an automated observation can prompt the dome to close.

It was not possible to achieve automation of the dome as part of this project because of the lack of a PC interface and concerns over possible equipment failure during remote operation. The following sections outline the requirements of dome automation, provide an overview of possible hardware solutions, and list possible methods of minimizing risk of equipment damage.

8.1. Current State of the System

The telescope system is housed in a fourteen-foot, six inch diameter dome, model REB manufactured by Ashdome. The dome sits on top of the optics shelter, forming the roof of a part of the building. The dome is able to rotate continuously in either direction. Dome rotation is powered by an electric motor. A user rotates the dome by pressing buttons on a control box that sits just below the rotating dome. The motor and rotation control panel are both mounted to the building and completely stationary. The dome control panel is shown in Figure 13.

The rotating dome has a slit composed of two shutters. The upper shutter swings back over the dome to open, and the lower shutter swings out to open. Both shutters are motorized, and a user controls both from a second control panel. The shutter control panel includes buttons for opening and closing each shutter. This control panel is mounted to the side of the dome and actually rides with the dome as it rotates. There are manual overrides for the motors as well. Figure 14 shows the shutter control panel.

Figure 13: Dome rotation controller and motor are mounted to the building and are stationary

Figure 14: Shutter control panel rides with the dome as it rotates

Figure 15: Lower shutter control motor and limit switch

Figure 16: Upper shutter motor

Both shutters have limit switches installed that tell the controllers when to stop the motor. These limit switches are activated when the shutter reaches a particular position, such as completely open or completely closed. Activating the limit switches stops the motor, which could burn out if it were to continue trying to move a shutter past its physical limits. In addition to the limit switches, there are a few other safety features built into the controllers. The buttons that move the shutters are on a delay, so a user needs to hold them for several seconds before the shutters move to ensure that the user meant to move the shutters. A stop button allows the user to stop the shutter from opening or closing instantaneously. The limit switches are also used to ensure that the lower shutter is completely closed before the upper shutter begins to close. This feature ensures that they shutters close properly.

The shutter controller and two motors that actually move the shutters are mounted to the dome, which moves relative to the stationary building. The two shutter control motors are shown in Figure 15 and Figure 16. In order to power the shutter motors and controller, a set of slip rings transfers electric power to an electrical outlet that is mounted to the movable dome, shown in Figure 17. The slip rings receive power from a metal contact for each of the three plug pins (hot, neutral, and ground). As the dome rotates, the slip rings maintain contact with the stationary power supply, shown in Figure 18. This way, power is available to any component that travels with the rotating dome.

Figure 17: Slip rings move with the dome to power mobile outlet

Figure 18: Slip rings move against stationary power supply

On top of each controller is a transceiver. While the project team did not have the opportunity to test them, these devices can be used to provide a data link between the controller boxes. According to the dome manual, which is required to remain inside the optics shelter, a remote control can be attached to the stationary control panel to allow control over dome rotation and the shutters from a single controller. In such a case, the transceivers would be used to transmit signals to open or close shutters. A wireless data link is necessary since the shutter control panel moves with the dome, and a cable would get tangled around the equipment as the dome rotates. The transceivers are shown in Figure 19.

Figure 19: Dome control transceivers

The dome already has in place most of the components that are necessary to automate its operation and allow remote control. The actuators to move the components, control panels, slip rings, and a remote control interface are provided with the dome system that is already in place. This equipment provides everything that is necessary to move dome components, and the only thing that must be added for an automated system is a method to obtain the current position and a PC interface.

In order to be able to move to the dome to a specific orientation, two sensors would be required. One must be a home position sensor. This sensor would be installed at one point along the dome's track and determine an absolute starting position. All dome movements would be relative to the home position. Keeping track of where the dome is relative to the home sensor requires counting the revolutions of the motor that drives it. To provide an automated system, these two sensors would need to be connected to a PC that can track the dome's operation.

The PC interface should provide two way communications: it must receive input from the home sensor and position counter, and it must output commands for how to move the dome to the remote control input of the control panels. The software that operates the dome should be able to position the slit to a given azimuth, and provide controls for opening and closing the shutters. Figure 20 provides a block diagram to summarize which components must be added for the dome to become automated.

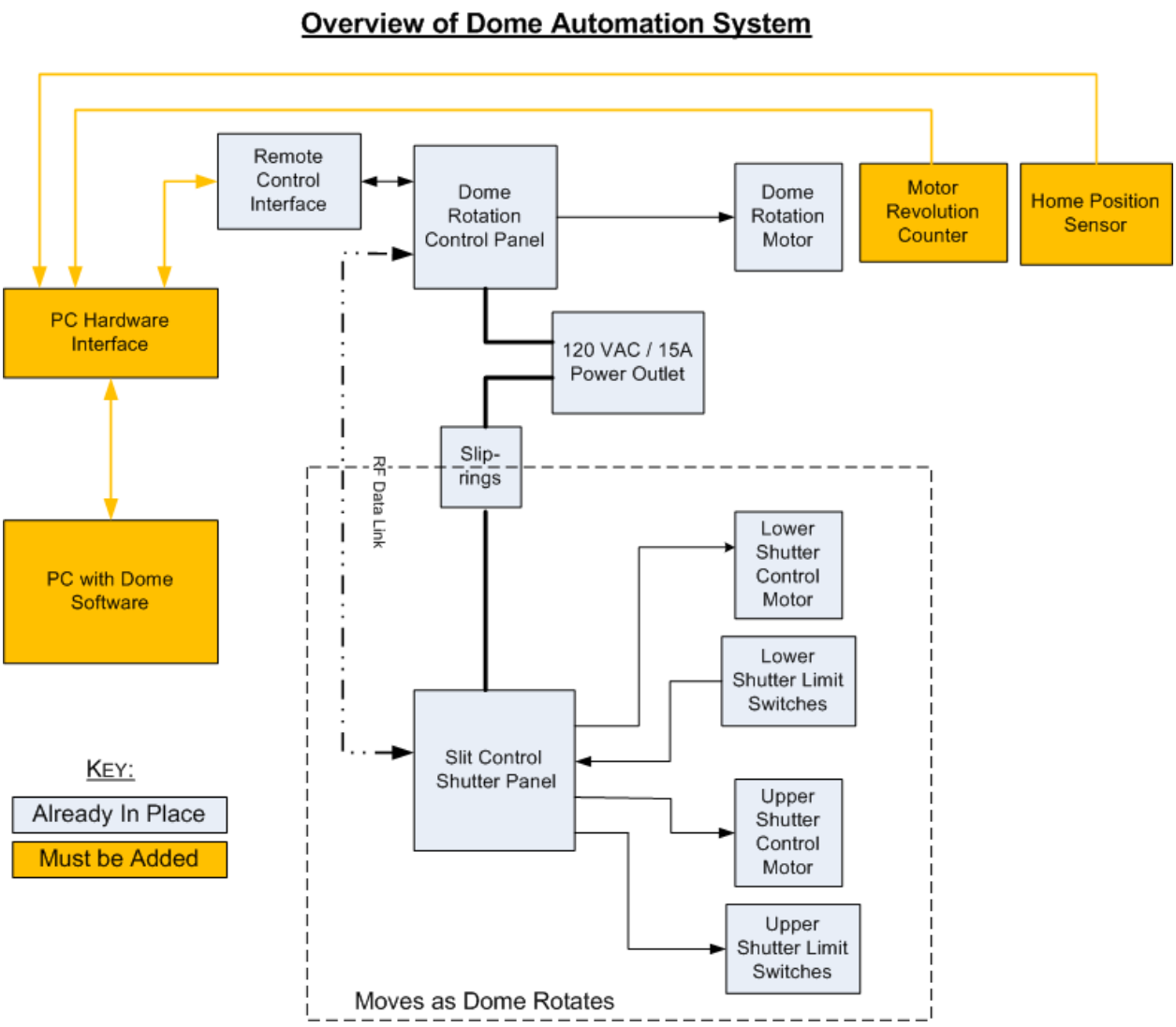


Figure 20: Block diagram of automated dome system

Once the hardware is in place, the best way to incorporate dome control software into the existing project would be through a software package called Automadome. This software package is from Software Bisque, which is the same company that manufactured the mount and its controlling software, TheSky6. Automadome would integrate well because like TheSky6, the program can be controlled with VBscripts that can be called by the back-end driver. Additionally, Automadome is designed to be completely compatible with TheSky6 so as to automatically orient the dome slit in the same direction as the telescope mount. Automadome costs $249, and more information is available on the Software Bisque website.

8.2. Comparison of Potential Solutions

The project team has researched and identified several potential solutions to complete automation of the dome. If the resources are available, then the best and least expensive

solution may be to design and implement a PC interface as described in the previous section. Most of the hardware necessary to implement dome automation is in place, and the remaining components are not excessively complex.

If it is not feasible to develop an interface at SRI or HAARP, then there are other options. Two of these solutions are consultants who upgrade and automate domes across the country. The third option is a kit that provides controller circuits to interface the dome components with a PC. This section provides a description of the solutions and discusses advantages and disadvantages of each. Referenced email correspondence can be found in Appendix M.

Astronomical Consulting:
This company was strongly recommended by Ashdome, the manufacturer of the dome. Richard Olsen of Ash Mfg. Co. stated that Astronomical Consulting has automated many Ashdomes across the country and that there should be no problem with automation of the specific dome that was installed at HAARP.

Astronomical Consulting stated that the company can only perform the installation during the summer months. The company did not provide an estimated price for the installation.

Contact information:
Dr. Peter Mack, President & C.E.O.
Telephone: 520-219-8722
Fax: 520-219-7989
Postal address: PO Box 91946, Tucson, AZ 85752-1946
Electronic mail: General Information: info@astronomical.com
Customer Support: support@astronomical.com

Meridian Controls:
This company specializes in computerized dome automation. The provided automation package is compatible with *Automadome* software, and the dome hardware interfaces with the PC over RS-232.

When the project team contacted Bryce Bennett, Ph.D. of the Royal Military College of Canada, the feedback was that his Meridian Controls installation was unreliable and he was not satisfied with it. Dr. Bennett added that he removed the Meridian Controls system and would replace with one from Astronomical Consulting.

In a telephone conversation, Meridian Controls provided an estimated cost of automating the dome. As actual specifications of the dome were not available at the time, the company quoted

automation of a non-motorized dome without installed slip rings. The quoted cost was $9000 for the system plus $1600 for travel and expenses.

Contact information:
Postal address: PO Box 839, 1085 2nd Street, Berthoud, CO 80513
Telephone: 877-295-9797
Fax: 530-325-7700

MaxDome II:
MaxDome II is a dome automation kit from a company called Diffraction Limited. The kit provides two controller cards: one card connects to a PC, the dome rotation motor, and dome rotation sensors, and the other card rides on the dome and connects to the shutter motors and limit switches. The two cards communicate via an RF signal. The system is compatible with *Automadome* software. Each controller card is designed to drive 12VDC motors up to 5A directly. However, since the motors used in the dome operate on AC power, relays would need to be installed to implement this system. The price of the MaxDome II with shutter and rotation control is $1395.

The project team contacted David Sonnek, who uses the MaxDome II system with an Ashdome in his personal observatory. David stated that the system is simple, but has been reliable. However, he noted that he only uses the system in a limited capacity – he does not operate it remotely and he does not use it to open and close the shutters. His opinion seems to be that MaxDome II is "is an easy way to glue together a control PC, sensors and the Ashdome motor" if the intention is merely to synchronize the dome orientation with that of the telescope. He implies that a more robust solution is necessary if the intention is to provide automated and remote control of the dome.

The project team's opinion is similar to that of Mr. Sonnek. While this system is simple and includes an Automadome compatible PC interface, it is restrictive and would limit the functionality of the installed hardware. Since this controller board must drive the motors directly, it would bypass the installed control panels. The team recommends a solution that adds remote capabilities to the existing on-site control, in which case MaxDome II is not the best solution.

Contact Information:
Diffraction Limited
Postal Addres: 100 Craig Henry Drive, Unit 202, Ottawa, Ontario K2G 5W3, Canada
Telephone: 613-225-2732
Website: http://www.cyanogen.com/products/dome_main.htm

8.3. Risk Mitigation

The remote and automated operation of a dome system presents some inherent risks to the dome and the equipment it shelters. However, with proper planning these risks can be minimized by ensuring that personnel intervene in case of a problem. The greatest risk is posed by precipitation, especially snow, which can cause costly damage to the telescope equipment. Other risks include jammed or frozen dome components, broken limit switches, and burned out motors. These events can cause damage to the dome and subsequently leave the telescope system exposed and vulnerable to weather damage.

In the last several years, a limit switch at the HAARP dome froze and caused the upper shutter motor to run until it burned out, which immobilized the shutters in an open position. Consequently, there is an appropriate level of concern over enabling remote control of the dome; however, proper precautions can help ensure that the equipment is protected.

The best way to mitigate the risk of operating the dome remotely is to install a webcam in the optics shelter. Cameras that connect to a computer via a USB cable are inexpensive and readily available. Such a camera should take an image of the inside of the dome every few seconds and make the image available over the internet. The Apache server that was installed as part of this project can allow a telescope operator at a remote site to see the conditions inside the dome. Specifically, the operator would be able to see whether there is precipitation entering the dome and whether the dome opens and closes properly, which would help prevent burned out motors. If the dome does not move on command, then the remote operator can contact on-site personnel for additional assistance.

Since the dome is opened in low-light conditions and nighttime, a night-vision webcam would be the best option. These devices detect infrared wavelengths that do not interfere with optical experiments at HAARP. Here are two products that may be implemented:

USB Qcam with NightVision:
The infrared USB webcam offers 1.3 megapixels of resolution and costs $35. More information is available at http://www.usbgeek.com/prod_detail.php?prod_id=0493

8-LED Infrared Night-Vision USB Webcam:
This infrared USB webcam has 640x480 video resolution and a metal body. It costs $18. More information is available at http://insidecomputer.stores.yahoo.net/6inniusb35we.html

In addition to verifying operation of the dome through a webcam, a remote user should be able to know the weather conditions on-site to determine whether it is safe to open the dome. The project team researched various weather stations that measure precipitation to find a solution. Most use a “tipping bucket” design that requires some precipitation to build up before any is

detected. While these devices can track precipitation accumulation accurately over time, they are not suited to detect precipitation and immediately notify a remote dome operator.

The best solution that the project team found for this problem is the Boltwood Cloud Sensor II from Diffraction Limited. This product is specifically designed for use by observatories to detect weather conditions that may pose a threat to equipment if the dome is open. This weather sensor can detect cloud cover, precipitation, and wind speed. This cloud sensor costs $1500. The product is shown in Figure 21.

Figure 21: Boltwood Cloud Sensor II from Diffraction Limited
(source: http://www.cyanogen.com/products/cloud_main.htm)

According to the manufacturer's website, this product measures cloud cover by comparing ambient ground temperature with the predicted temperature of the sky, which is determined by measuring infrared radiation. Cloud cover is a good indicator of when to open or close the dome as incoming clouds can foretell precipitation. Additionally, an observatory would not be able to see anything through clouds, so there is no reason to have the dome open and the equipment exposed if clouds are present.

The Boltwood Cloud Sensor II includes a heated moisture sensor that detects rain or snow. The project team contacted the manufacturer to ask whether the product would be able to work in the harsh Alaskan climate, and the company was reassuring that this product has worked well in such conditions. The product also includes an anemometer with no moving parts.

The cloud sensor interfaces with a PC through a USB port. All information gathered by the instrument is displayed on the screen, and the user can set thresholds for alerts. The included software has ActiveX controls built in, which allows the program to be incorporated into other software. The user interface is shown in Figure 22.

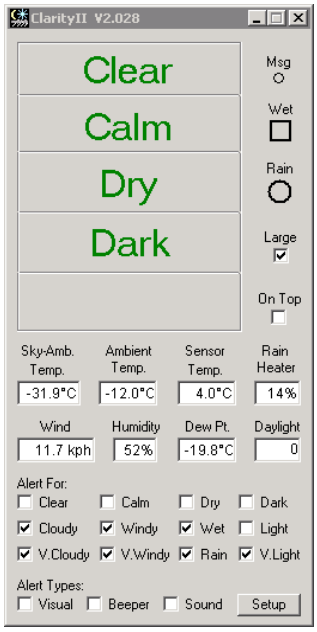


Figure 22: Boltwood Cloud Sensor II user interface
(source: http://www.cyanogen.com/products/cloud_main.htm)

Additionally, the cloud sensor has an output that can be interfaced with a dome controller in addition to the PC. In this case, any weather conditions that trigger an alert in the user interface would also instruct the dome controller to close the shutters immediately. This method of protecting the equipment from the weather may be even more reliable than waiting until someone on site becomes aware of the weather and goes to the optics shelter to shut the dome manually.

The project team recommends the installation of the cloud sensor to alert remote users of potentially threatening weather conditions. Additionally, the implementation of a webcam inside the dome would minimize the risk of damage as a result of the failure of a dome component. With these safety factors in place, the dome at the HAARP site can be automated safely to give remote users full control of the observatory.

9. Conclusion

The goal of this project was to develop a system to allow remote control of the optical imaging system at HAARP. The remote control system was designed with the intention of allowing researchers at SRI the ability to gather data without traveling to Alaska merely to run the equipment. The system needed to provide users with the ability to open the dome that houses the system, orient the mount, specify settings for each observation session, and capture images autonomously.

The project team was able to develop a web-based interface that allows control of the mount, and both sets of imaging equipment. The system is operational and has been set up to run on the computers that control the hardware. The data that is gathered during each observation session is stored on the computers at HAARP, and an installation of DeltaCopy sends these files to a computer in California autonomously. A permanent Linux server for these image files must be installed at SRI to make the images easily accessible to researchers. Then, the DeltaCopy installation on-site can be reconfigured to upload to the new server. The web-friendly PNG images that are created during each observation are stored on the on-site computer as well. The directories that hold them are accessible to the internal HAARP network and the HAARP public web server. A web interface should be created to allow access to these images from the public HAARP website. A remaining issue with the PNG images only is that the image processing algorithm distorts some images from the wide-angle camera if they are acquired before twilight. An adjustment to the “5-95” algorithm would resolve the problem. Otherwise, the remote control system has been shown to the stable and reliable for controlling the equipment and gathering data.

The project team was not able to automate the dome due to safety concerns. However, this report presents several solutions for enabling remote control of the dome in a safe manner. The addition of a weather detector and webcams allows the remote user to remain aware of weather conditions on-site and provides verification that the dome operates properly. Most of the components that are needed to automate dome operation are in place, and all that is needed is a PC interface and position sensors. The project team recommends the addition of these components to complete the automation of the observatory.

The project team had the opportunity to travel to Alaska and finalize development of the system on-site. At the conclusion of the trip, the team stood by as the system was actually tested by a researcher at SRI. The system was able to gather real data and operate reliably. The few minor bugs that were noticed were corrected upon return to SRI. The remote control system can now be used to control the equipment and collect data.

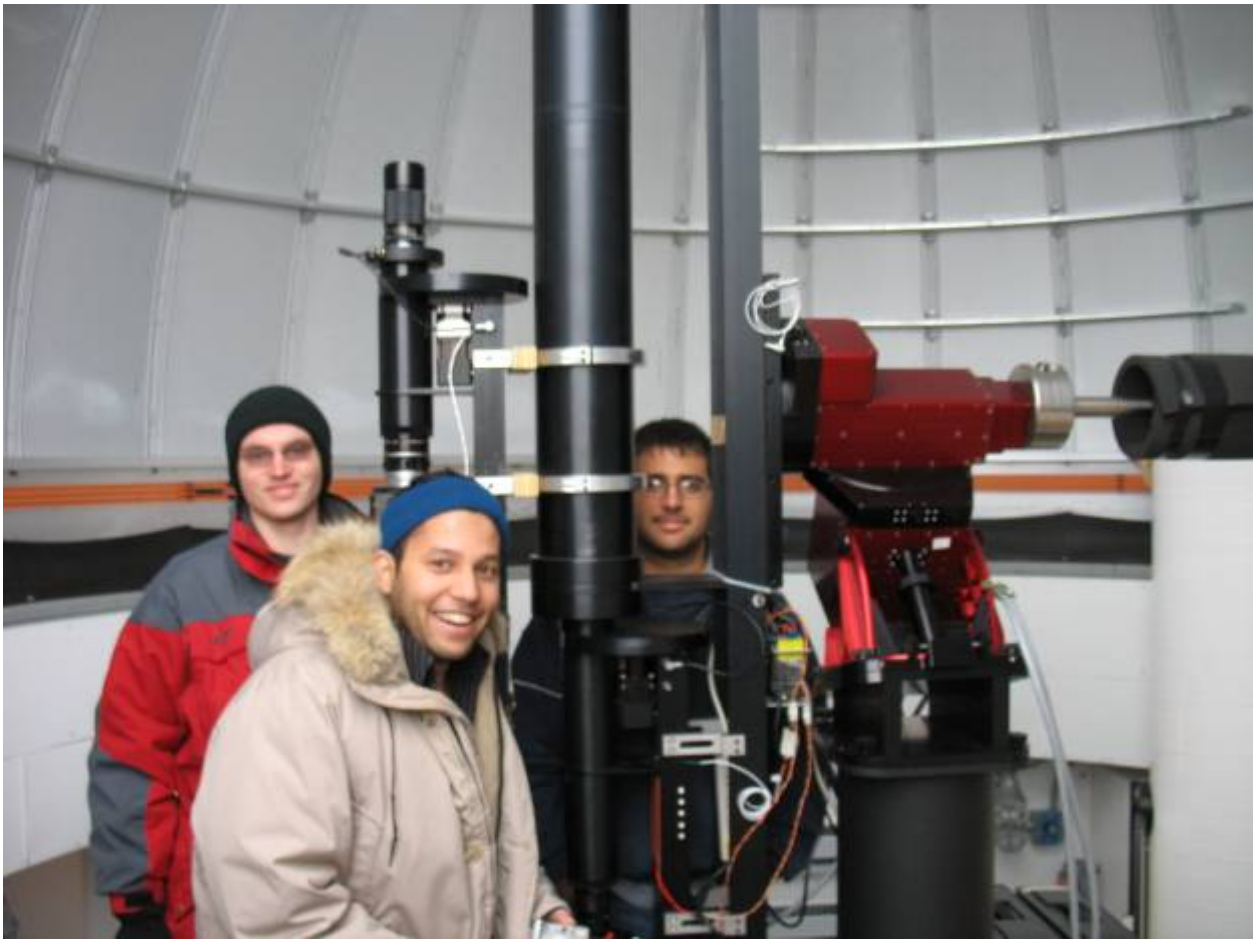


Figure 23: The project team on-site with the optical imaging system

10. References

Apache HTTP Server Version 2.2 Documentation. The Apache Software Foundation. 2008. 5 Mar. 2008. <http://httpd.apache.org/docs/2.2/>

Beazley, David M. Python Essential Reference. Indianapolis: Developer's Library, July 2007.

"The Boltwood Cloud Sensor II." Diffraction Limited. 2005. 5 Mar. 2008. <http://www.cyanogen.com/products/cloud_main.htm>

The High Frequency Active Auroral Research Program. 2007. HAARP. 5 Mar. 2008. <http://www.haarp.alaska.edu/haarp/index.html>

"MaxDome II Observatory Dome Control System." Diffraction Limited. 2004. 5 Mar. 2008. <http://www.cyanogen.com/products/dome_main.htm>

Microsoft Developer Network. 2008. Microsoft Corporation. 5 Mar. 2008. <http://msdn2.microsoft.com/en-us/default.aspx>

Princeton Instruments. Princeton Instruments VersArray System. Trenton, NJ: Roper Scientific, Inc., 2004.

Princeton Instruments. PVCAM 2.7. Trenton, NJ: Roper Scientific, Inc., 2004.

Thompson, Chris. "Windows + Apache 2.0 + SSL." Thompsonbd. Version 1.2. 5 Mar. 2008. <http://www.thompsonbd.com/tutorials/apachessl.php>

Trondsen, Trond S. Narrow-Field Auroral Imager for Northwest Research Associates, Inc. Keo Scientific, Ltd., January 2006.

Software Bisque. 2007. Software Bisque. 5 Mar. 2008. <http://www.bisque.com/>

Synametrics Technologies. "DeltaCopy User's Guide." About my X. <http://www.aboutmyip.com/files/DeltaCopyManual.pdf>

SRI International. 2008. SRI International. 5 Mar. 2008. <http://www.sri.com>

Vandevenne, Lode. "LodePNG for C (ISO C90) and for C++." LodePNG. 2008. 5 Mar. 2008. <http://members.gamedev.net/lode/projects/LodePNG/>

W3 Schools. 2008. Refsnes Data. 5 Mar. 2008. <http://www.w3schools.com/>

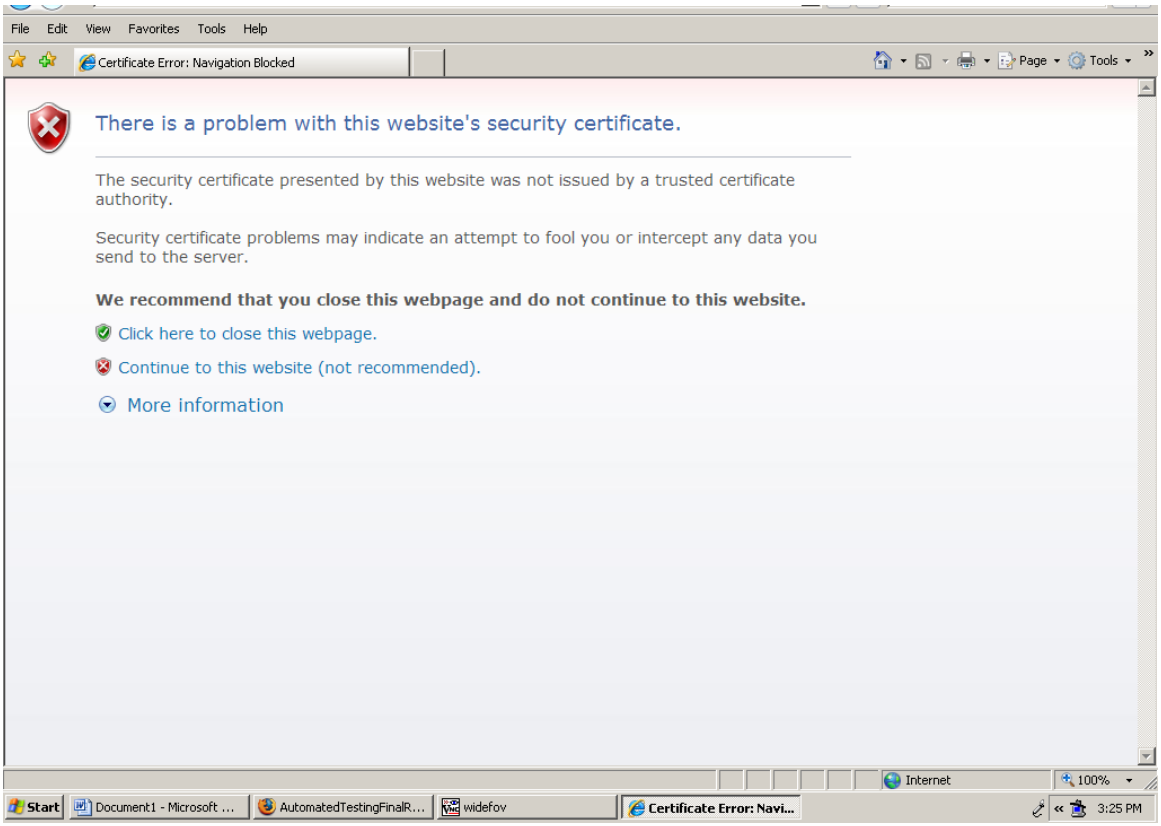
Appendix A: User's Guide

Step 1:

Open up a web browser and enter https://***.***.**.**:8000/ in the URL. Be sure to use *https* and not *http*.

Step 2:

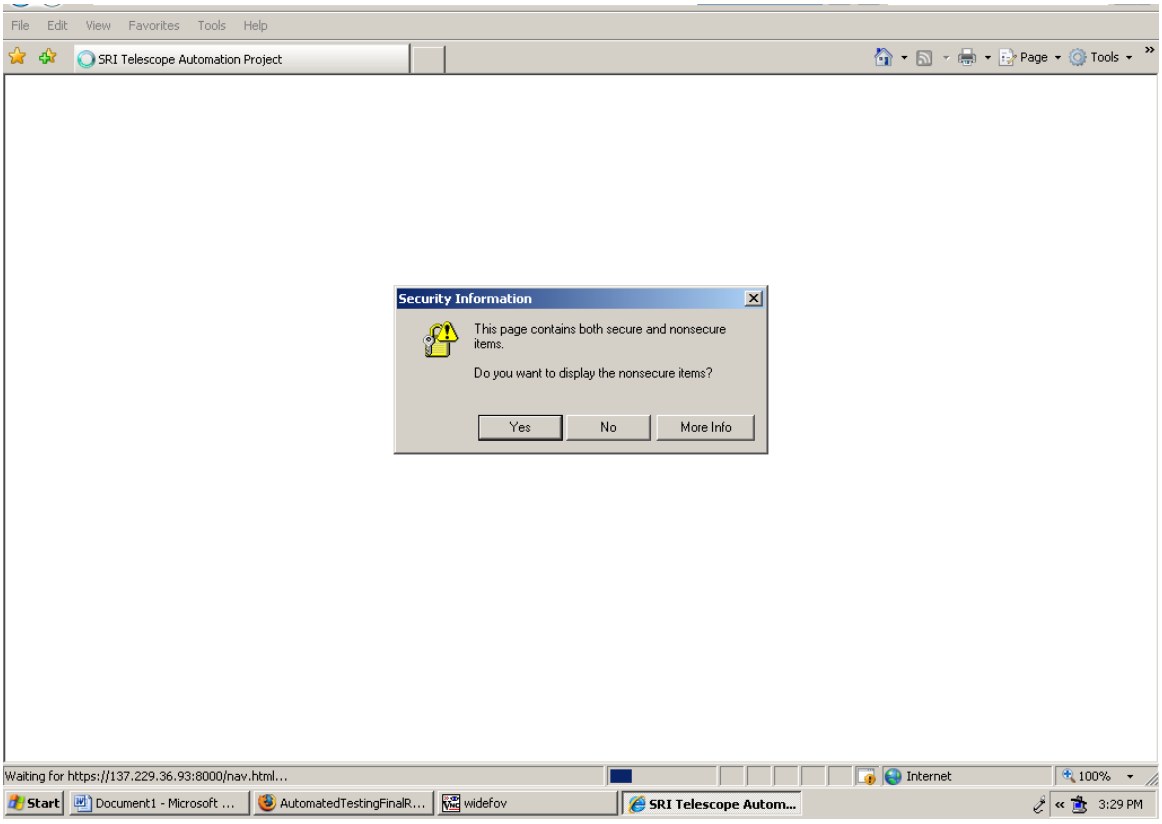
In Internet Explorer, the following page will show up:



Click on the second option "**Continue to this website (not recommended)**". The browser throws this error because the security certificate generated for the website was not issued by an issuing authority. The certificates were generated at SRI and are valid, so it is safe to proceed.

Step 3:
The next page will prompt for a user name and a password as shown below. Enter the provided username and password. Click on **OK**.

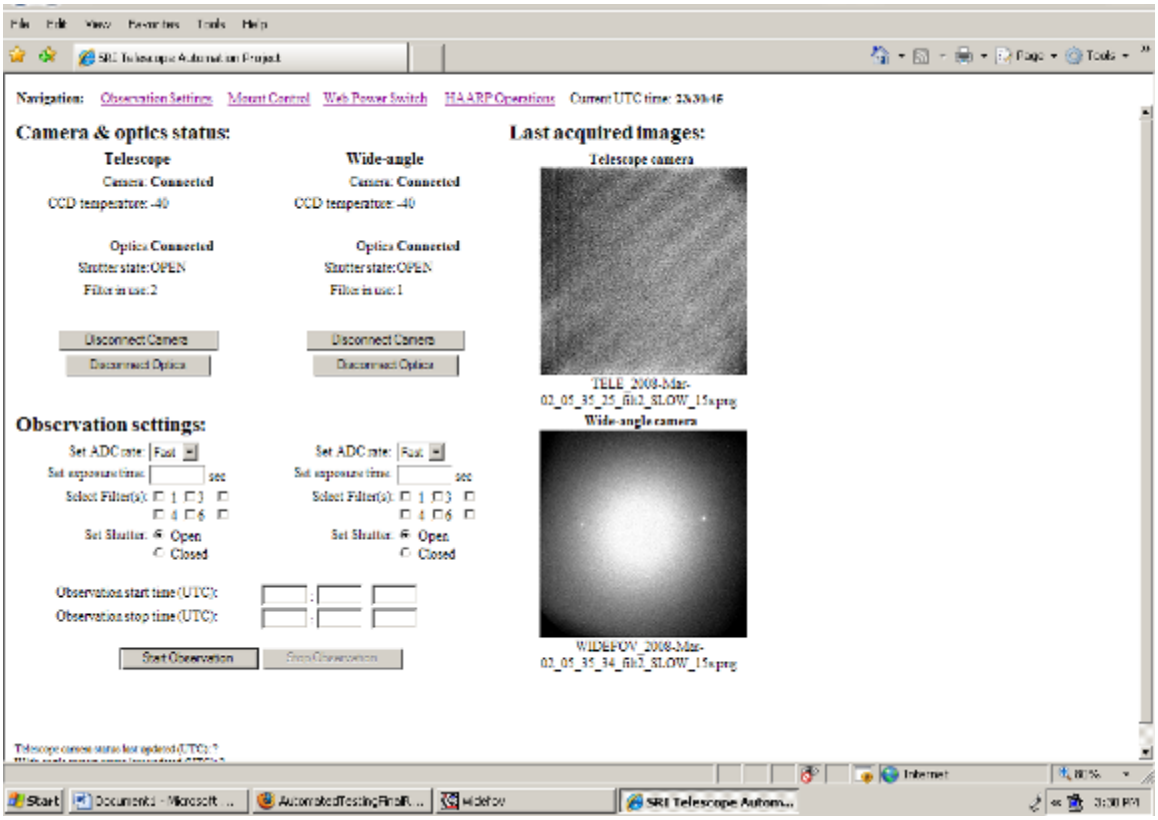
Step 4:
The following page shows up next:



Click on **Yes**. This error shows up because the UTC clock that displays at the top of the page is provided by a third-party website. If it is felt that this feature poses a security risk, then it may be removed by editing the *nav.html* file.

Step 5:

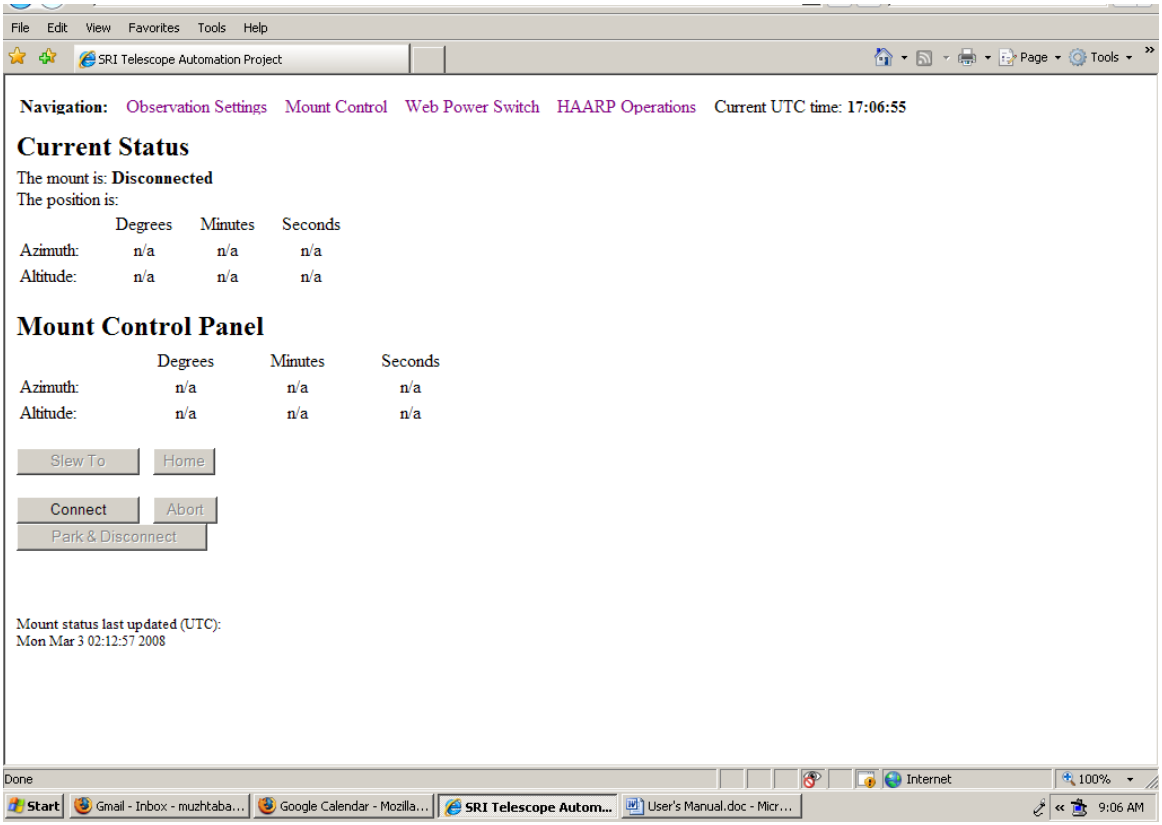
On successfully entering the user name and password, the following page shows up (the actual images may be different):



This is the home page that loads up when a user first logs in. The page defaults to the Observation Settings page on the top panel. Some of the other pages available for navigation on the top panel are Mount Control, Web Power Switch and HAARP Operations.

The **Observation Settings** page allows a user to see as well as specify settings for both the wide-angle and the telescope cameras and optics. The top half of the page lets the user know of the status of the hardware, which is updated periodically. The bottom half of the page allows the user to specify settings such as the ADC rate, the exposure time, the filter(s) to use and the state of the external shutter. In addition to this, a start and a stop time may also be included if the user wants to carry out an observation at a later time. All the time fields have to be entered in UTC time.

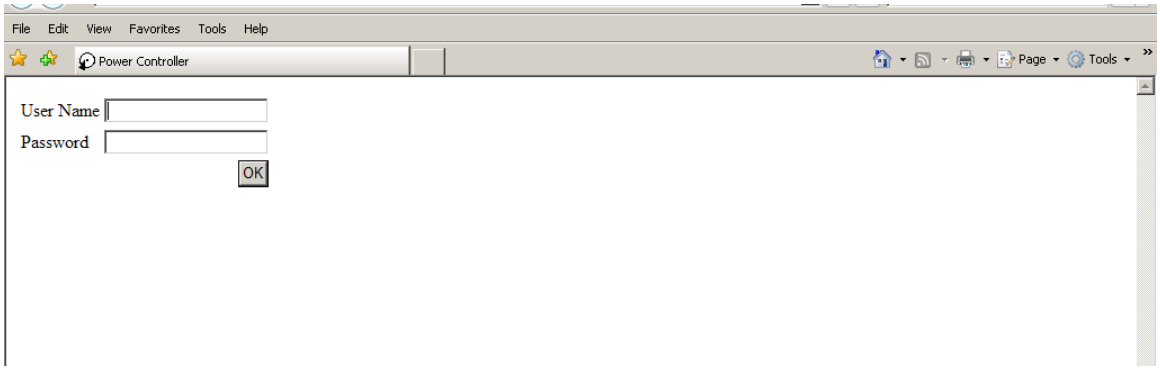
The following shows the **Mount Control** page when a user loads it up initially:



This page allows the user to see and specify the mount settings. The top half of the page displays the current status of the mount hardware. It tells the user if a mount is connected and the current coordinates of its position. The bottom half of the page allows the user to connect to the mount, specify coordinates to slew to, park and disconnect the mount or abort an operation in progress. The coordinates for the mount position have to be specified in degrees, minutes and seconds for both azimuth and altitude.

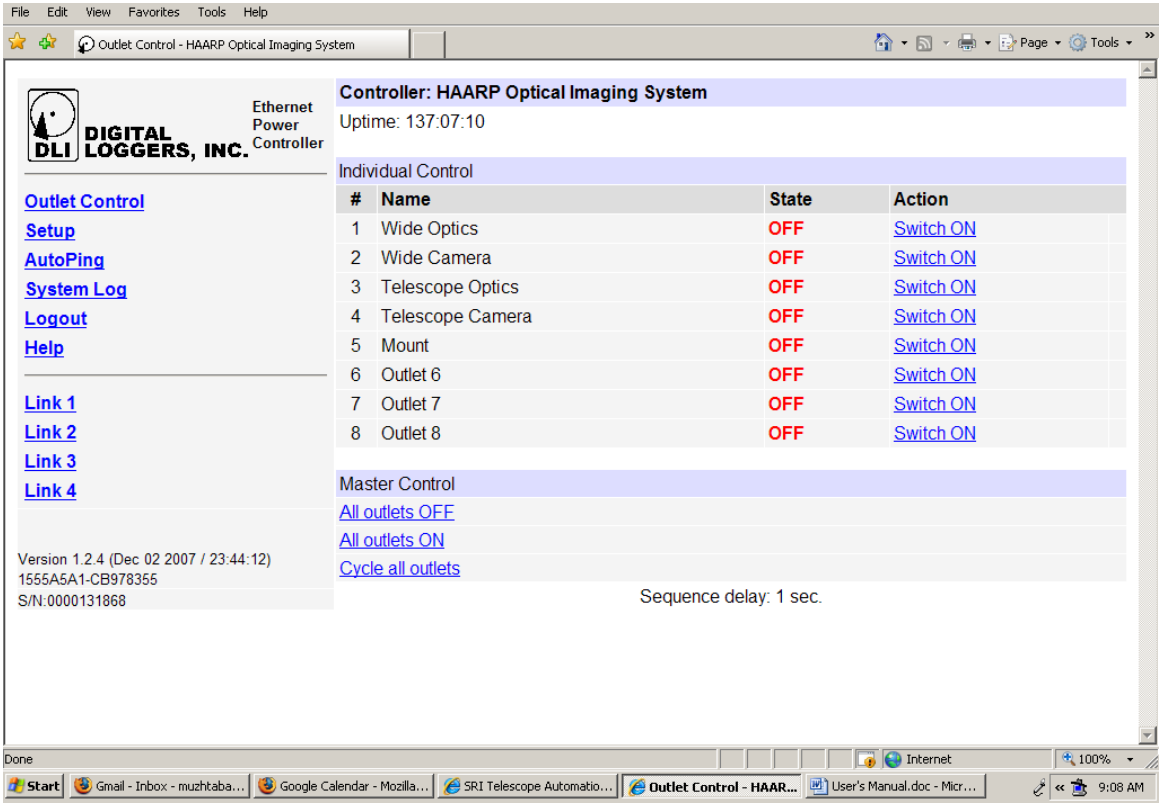
Important: The Degrees, Minutes and Seconds fields cannot take on empty values!

On clicking the **Web Power Switch** link, the following page loads up in a new window, prompting the user for a user name and a password:



Enter the provided username and password.

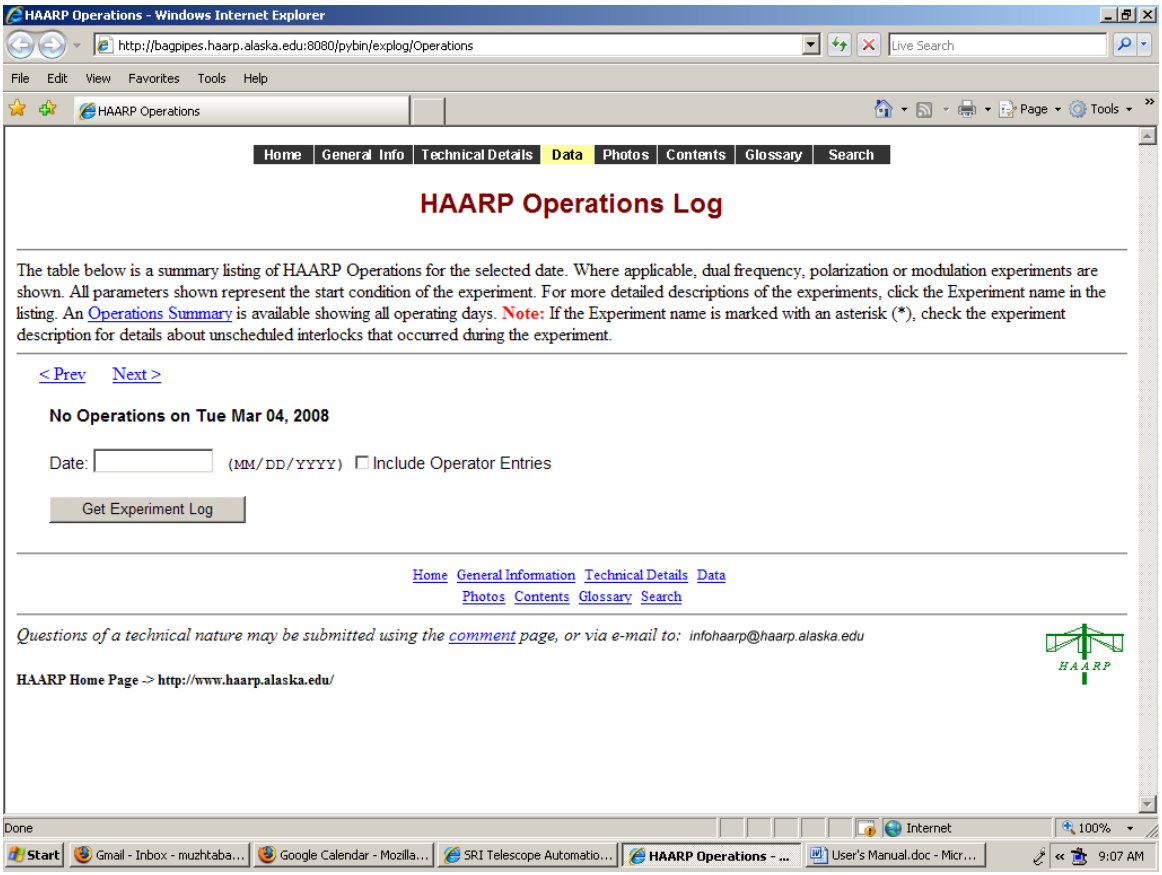
On successfully entering the user name and password, the following page shows up:



This page allows the user to power up or power down the set of cameras and optics as well as the mount. The user can choose to power up any of these hardware devices by clicking on the corresponding **SWITCH ON** button or power down any of these hardware devices by clicking on the corresponding **SWITCH OFF** button. The current state of the hardware is displayed under the **STATE** field. The **All outlets OFF** button under **Master Control** allows the user to power

down all the hardware equipment and the **All outlets ON** button under **Master Control** allows the user to power up all the hardware equipment. Once the user has finished powering the equipment on or off, the user may safely exit the page using the **Logout** link on the left plane.

The **HAARP Operations** page shown below may be used to check the schedules of currently planned observations by the scientists at HAARP.



Appendix B: Maintenance Guide

The software that has been written for the project resides on one of the computers at HAARP in Alaska. The software was written keeping scalability and future expansion in mind. Since most of the software is based in Alaska, remote maintenance of the code is of paramount importance, and some features were incorporated into the system accordingly.

First, we have chosen to provide remote access to both the computers at HAARP from the SRI location. The freeware version of TightVNC may currently be used to remotely log in into one or both of the computers up there, for troubleshooting purposes. The VNC servers up there have been configured to allow remote logging in from thus allowing people on the SRI domain and who have the appropriate user name and password to log in. Remote logging is useful to check on the status of the background driver or to see if the raw image data is actually being backed up locally. Short of the server at HAARP actually freezing or crashing, the VNC feature allows users to access the computers just as if they were on site. If the backend driver program crashes, the user could VNC into the computer, use the task manager to end the backend driver process, and manually start the program. If the backend driver program crashes on the wide angle machine, the user will have to go to the "C:\Program Files\Telescope Automation\Apache\htdocs\wide" directory and double click the BackendDriver program there in order to manually restart the program. If the crash happens on the telescope machine, the user will have to go to the Z:\ share drive under "My Computer" and double click the BackendDriver program there in order to manually restart the program on the telescope machine. It will be apparent if the back-end driver is not operating as the website will display "Unable to reach IP address!" when the user tries to submit commands.

If the on-site computers cannot be reached by VNC, it is likely that the entire system may have frozen. In this case, someone on-site must manually restart the computer. An alternate solution is to connect the computers to the web power switch to allow remote restarting. In such a situation, the user would have to enter the IP address http://***.***.**.**:8001/ in the URL of the browser. This would take the user directly to the **Web Power Switch** control page, as the web power switch operates independently of the computers.

The other maintenance feature that has been incorporated into the back-end driver is the Telnet functionality. This allows for debugging of individual commands and faster troubleshooting. If a developer wants to test a single command being sent to the backend driver over TCP/IP, the developer could open up a Telnel console and connect to the IP address of the server over port 5000. The developer could then issue single commands and check to see what the driver responds with. The user must access the Telnet functionality from within the HAARP network, perhaps by using VNC to access the on-site computers.

Appendix C: Directory Paths of System Files

This appendix lists each of the subdirectories that contain project files. Please refer to these lists if replacing any files that are a part of the remote control system.

C:\Program Files\Telescope Automation\Apache\cgi-bin:
control.py

C:\Program Files\Telescope Automation\Apache\htdocs:
<dir> Tele
<dir> Wide
default_image.png
imBig.js
index.html
layout.css
mntUserPrefs.xml
mount.html
mount.js
nav.html
observation.html
observation.js
obsTelePic.html
obsUserPrefs.xml
obsWidePic.html

C:\Program Files\Telescope Automation\Apache\htdocs\tele:
<dir> png
<dir> winview
BackendDriver.exe
default_image.png
telecamStatus
winview_header

C:\Program Files\Telescope Automation\Apache\htdocs\wide:
<dir> mount
<dir> png
<dir> winview
BackendDriver.exe
default_image.png
mountStatus.xml
wacamStatus.xml
winview_header (binary file)

C:\Program Files\Telescope Automation\Apache\htdocs\wide\mount:
abort
abort.vbs
connect
connect.vbs
coupledome.vbs
disconndome.vbs
getposition
getposition.vbs
home
home.vbs
iscomplete.vbs
isconnected.vbs
park
park.vbs
slewto
slewto.vbs

Appendix D: HTML and Javascript Files

layout.css

```
h2 {padding-bottom: 5px; margin-bottom: 0;}
h3 {padding-bottom: 0; margin-bottom: 0;}
div {padding: 0px; margin: 0px; display: inline;}
form {margin: 0px; padding: 0px; display: inline;}
.button {padding: 0px; margin: 2px;}
```

index.html

```
<html>
  <head>
    <title>SRI Telescope Automation Project</title>
  </head>
        <frameset rows="35,*" border="0">
                <frame name="navbar" src="nav.html" scrolling="no">
                <frame name="main" src="observation.html">
        </frameset>

</html>
```

mount.html

```
<html>
 <head>
        <link rel="stylesheet" type="text/css" href="layout.css">
        <script src="mount.js" type="text/javascript"> </script>
 </head>

<body onload=loadXML()>


        <!-- Current state of the system from XML -->
        <h2>Current Status</h2>

        <div id='errorDisplay'></div>

        The mount is: <b> <div id='statusDisplay'> </div> </b>
        <br>

        The <div id="posUpd"> </div> position is:
        <table width="300" border="0">
                <colgroup>
                        <col>
                        <col align="center">
                        <col align="center">
                        <col align="center">
                </colgroup>
                <tr>
                        <td> </td>
                        <td>Degrees</td>
                        <td>Minutes</td>
                        <td>Seconds</td>
                </tr>
                <tr>
                        <td width="75">Azimuth: </td>
                        <td width="75"> <div id="aziDegDisplay"> </div> </td>
                        <td width="75"> <div id="aziMinDisplay"> </div> </td>
                        <td width="75"> <div id="aziSecDisplay"> </div> </td>
                </tr>
                <tr>
                        <td>Altitude: </td>
                        <td> <div id="altDegDisplay"> </div> </td>
                        <td> <div id="altMinDisplay"> </div> </td>
                        <td> <div id="altSecDisplay"> </div> </td>
                </tr>
```

```
	</table>



	<!-- User controls form -->
	<h2>Mount Control Panel</h2>

	<form method="POST" action="/cgi-bin/control.py" name="mountSlewForm" onsubmit="return
validate_form()">
		<input type="hidden" name="equipment" value="mount">
		<input type="hidden" name="mntOp" value="slew">

		<table width="400" border="0">
			<colgroup>
				<col>
				<col align="center">
				<col align="center">
				<col align="center">
			</colgroup>
			<tr>
				<td> </td>
				<td>Degrees</td>
				<td>Minutes</td>
				<td>Seconds</td>
			</tr>
			<tr>
				<td width="100">Azimuth: </td>
				<td width="100"> <div id="setAziDegDisp"> </div> </td>
				<td width="100"> <div id="setAziMinDisp"> </div> </td>
				<td width="100"> <div id="setAziSecDisp"> </div> </td>
			</tr>

			<tr>
				<td>Altitude: </td>
				<td> <div id="setAltDegDisp"> </div> </td>
				<td> <div id="setAltMinDisp"> </div> </td>
				<td> <div id="setAltSecDisp"> </div> </td>
			</tr>
		</table>

		<br>
		<input name="submit" type="submit" value="    Slew To    " id="slewButton">
	</form>

	<form method="POST" action="/cgi-bin/control.py">
		<input type="hidden" name="equipment" value="mount">
		<input type="hidden" name="mntOp" value="home">
		<input name="submit" type="submit" value="Home" id="homeButton">
	</form>

		<br><br>

	<form method="POST" action="/cgi-bin/control.py">
		<input type="hidden" name="equipment" value="mount">
		<input type="hidden" name="mntOp" value="connect">
		<input name="submit" type="submit" value="    Connect    " id="connectButton">
	</form>

	<form method="POST" action="/cgi-bin/control.py">
		<input type="hidden" name="equipment" value="mount">
		<input type="hidden" name="mntOp" value="abort">
		<input name="submit" type="submit" value=" Abort " id="abortButton"><br>
	</form>

	<form method="POST" action="/cgi-bin/control.py">
		<input type="hidden" name="equipment" value="mount">
		<input type="hidden" name="mntOp" value="park">
		<input name="submit" type="submit" value=" Park & Disconnect"
id="disconnectButton">
	</form>
```

```
        <br><br><br><br>
        <p><small>Mount status last updated (UTC): <div id="mountUpdated"> </div></small></p>

 </body>
</html>
```

nav.html

```
<html>

<table border="0">
        <tr>
         <td size="100"> <b>Navigation:   </b> </td>
         <td size="100"> <a href="observation.html" target="main">Observation Settings</a>
   </td>
         <td size="100"> <a href="mount.html" target="main">Mount Control</a>    </td>
         <td size="100"> <a href="http://***.***.**.**:8001" target="_blank">Web Power Switch</a>
   </td>
         <td size="100"> <a href="http://bagpipes.haarp.alaska.edu:8080/pybin/explog/Operations"
target="_blank">HAARP Operations</a>    </td>
         <td size="200" align="right"> Current UTC time: </td>
         <td size="100"> <iframe
src="http://free.timeanddate.com/clock/ix2xq66/fn15/ahl/avt/ftb/th1" frameborder="0" width="54"
height="19"></iframe></td>
        </tr>
</table>

</html>
```

observation.html

```
<html>
 <head>
        <link rel="stylesheet" type="text/css" href="layout.css">
        <meta http-equiv="REFRESH" content="900">
        <script src="observation.js"> </script>
 </head>

 <body onload=loadXML()>

        <table width="850" border="0" cellpadding="0">
                <colgroup>
                        <col align="right" valign="top"> </col>
                        <col align="left" valign="top"> </col>
                        <col align="right" valign="top"> </col>
                        <col align="left" valign="top"> </col>
                        <col align="center" valign="top"> </col>
                </colgroup>


                <!-- row of headers for page -->
                <tr valign="center">
                        <td width="550" colspan="4" align="left"> <h2>Camera & optics status:</h2>
<div id="teleErrorDisplay"></div> <div id="waErrorDisplay"></div></td>
                        <td width="300" align="left"> <h2>Last acquired images:</h2> </td>
                </tr>


                <!-- row of telescope and wide-angle headers, plus the last taken images -->
                <tr>
                        <td width="275" colspan="2" align="center"> <h3>Telescope</h3> </td>
                        <td width="275" colspan="2" align="center"> <h3>Wide-angle</h3> </td>
                        <td width="300" rowspan="10" align="center"> <b>Telescope camera</b> <br>
                                                        <a
href="javascript:void(window.open('obsTelePic.html','_blank','toolbar=no,location=no,status=no,di
rectories=no,menubar=no,scrollbars=yes,width=530,height=575,resizable=yes'))">
                                                        <img width="230" height="230" border="0"
name="teleCamLastImg"></a><br>
                                                        <div id="teleImgInfo">   </div> </td>
                </tr>
```

```
		<!-- rows of status info -->
		<tr>
			<td width="150"> Camera: </td>
			<td width="125"> <b><div id="teleCamStatus"> </div></b> </td>	<!--
status of telescope camera -->

			<td width="150"> Camera: </td>
			<td width="125"> <b><div id="waCamStatus"> </div></b> </td>	<!--
status of wide-angle camera -->
		</tr>
		<tr>
			<td> CCD temperature: </td>
			<td> <div id="teleCamCCDtemp"> </div> </td>
	<!-- telescope camera ccd temperature -->

			<td> CCD temperature: </td>
			<td> <div id="waCamCCDtemp"> </div> </td>
	<!-- wide-angle camera ccd temperature -->
		</tr>

		<tr>
			<td colspan="4">   </td>
	<!-- empty row to space out the content -->
		</tr>

		<tr>
			<td> Optics: </td>
			<td> <b><div id="teleOptStatus"> </div></b> </td>	<!-- status of
telescope optics -->

			<td> Optics: </td>
			<td> <b><div id="waOptStatus"> </div></b> </td>		<!-- status of
wide-angle optics -->
		</tr>

		<tr>
			<td> Shutter state: </td>
			<td> <div id="teleOptShutter"> </div> </td>		<!-- status of
telescope shutter -->

			<td> Shutter state: </td>
			<td> <div id="waOptShutter"> </div> </td>		<!-- status of
wide-angle shutter -->
		</tr>

		<tr>
			<td> Filter in use:	</td>
			<td> <div id="teleOptFilter"> </div> </td>		<!-- telescope
filter that is currently in use -->

			<td> Filter in use: </td>
			<td> <div id="waOptFilter"> </div> </td>		<!-- wide-
angle filter that is currently in use -->
		</tr>

		<tr>
			<td colspan="4">   </td>					<!--
empty row to space out the content -->
		</tr>

		<tr>
			<td colspan="2" align="center" valign="middle">
				<form name="teleCamForm" method="post" action="/cgi-
bin/control.py"> <!-- /cgi-bin/control.py -->
					<input type="hidden" name="equipment"
value="teleCam">
					<div id="teleCamConn"> </div>
				</form>				<!-- telecamera
connect or disconnect button is added by javascript -->
				<br>
```

```
                                        <form name="teleOptForm" method="post" action="/cgi-
bin/control.py">
                                                <input type="hidden" name="equipment"
value="teleOpt">
                                                <div id="teleOptConn"> </div>
                                        </form> </td>                 <!-- teleoptics connect or
disconnect button is added by javascript -->

                        <td colspan="2" align="center" valign="middle">
                                        <form name="waCamForm" method="post" action="/cgi-
bin/control.py">
                                                <input type="hidden" name="equipment"
value="wideCam">
                                                <div id="waCamConn"> </div>
                                        </form>                                         <!-- wacamera
connect or disconnect button is added by javascript -->
                                        <br>
                                        <form name="waOptForm" method="post" action="/cgi-
bin/control.py">
                                                <input type="hidden" name="equipment"
value="wideOpt">
                                                <div id="waOptConn"> </div>
                                        </form> </td>                 <!-- waoptics connect or
disconnect button is added by javascript -->
                </tr>

                <tr>
                        <td colspan="4">   </td>                                           <!--
empty row to space out the content -->
                </tr>
        </table>


        <form method="POST" action="/cgi-bin/control.py" name="obsForm" onsubmit="return
validate_form()">
        <table width="850" border="0" cellpadding="0">
                <colgroup>
                        <col align="right" valign="top"> </col>
                        <col align="left" valign="top"> </col>
                        <col align="right" valign="top"> </col>
                        <col align="left" valign="top"> </col>
                        <col align="center" valign="top"> </col>
                </colgroup>

                <!-- row of observation settings header -->
                <tr>
                        <td width="550" colspan="4" align="left"> <h2>Observation settings:</h2>
</td>
                        <td width="300" rowspan="9" align="center">
                                                                                <b>Wide-angle
camera</b> <br>
                                                                                <a
href="javascript:void(window.open('obsWidePic.html','_blank','toolbar=no,location=no,status=no,di
rectories=no,menubar=no,scrollbars=yes,width=530,height=575,resizable=yes'))">
                                                                                <img width="230"
height="230" border="0" name="waCamLastImg"></a><br>
                                                                                <div id="waImgInfo">
  </div>
                </tr>

                <!-- rows of observation setting options -->
                        <tr>
                                <td width="150"> Set ADC rate: </td>
                                <td width="125"> <div id="teleObsADCrateDisp"> </div> </td>

                                <td width="150"> Set ADC rate: </td>
                                <td width="125"> <div id="waObsADCrateDisp"> </div> </td>
                        </tr>

                        <tr>
                                <td> Set exposure time: </td>
```

```
                        <td> <div id="teleObsExpTimeDisp"> </div> </td>

                        <td> Set exposure time: </td>
                        <td> <div id="waObsExpTimeDisp"> </div> </td>
                </tr>

                <tr>
                        <td> Select Filter(s): </td>
                        <td> <div id="teleObsFilterDisp"> </div> </td>

                        <td> Select Filter(s): </td>
                        <td> <div id="waObsFilterDisp"> </div> </td>
                </tr>

                <tr>
                        <td> Set Shutter: </td>
                        <td> <div id="teleObsShutterDisp"> </div> </td>

                        <td> Set Shutter: </td>
                        <td> <div id="waObsShutterDisp"> </div> </td>
                </tr>

                <tr>
                        <td colspan="4">   </td>               <!-- empty row to
space out the content -->
                </tr>

                <tr>
                        <td colspan="2" align="center">Observation start time (UTC): </td>
                        <td colspan="2" align="left"> <div id="obsStartTimeDisp"> </div>
</td>
                </tr>

                <tr>
                        <td colspan="2" align="center">Observation stop time (UTC): </td>
                        <td colspan="2" align="left"> <div id="obsStopTimeDisp"> </div>
</td>
                </tr>

                <tr>
                        <input type="hidden" name="equipment" value="observation">
                        <td colspan="4" align="center"> <div id="teleSendToField"> </div>
                                                                                <div
id="waSendToField"> </div>
                                                                                <div
id="obsStartOp"> </div> <br>
                                                                                <input
type="submit" name="submit" value="Start Observation" id="obsStartButton">
                                                                                <input
type="submit" name="submit" value="Stop Observation" id="obsStopButton">   </td>
                        </tr>
        </form>

        <!-- disable the buttons until the status of the observation loads -->
        <script type="text/javascript">
                document.getElementById("obsStartButton").disabled = true;
                document.getElementById("obsStopButton").disabled = false;
        </script>

</table>


<br><br><br><br>
<small> Telescope camera status last updated (UTC): <div id="teleCamUpdated"> </div> <br>
                Wide-angle camera status last updated (UTC): <div id="waCamUpdated">
</div> </small>
 </body>

</html>
```

obsTelePic.html

```
<html>
	<head>
		<title>Telescope camera image</title>
		<link rel="stylesheet" type="text/css" href="layout.css">
		<meta http-equiv="REFRESH" content="900">
		<script src="imgBig.js"> </script>
	</head>

	<body align="center" onload=imgProc("tele")>
		<h3 align="center">Telescope Camera</h3>
		<img width="512" height="512" name="teleCamLastImg" alt="last telescope camera
image...">
		<br>
		<div id="filename"> </div>
	</body>
</html>
```

obsWidePic.html

```
<html>
	<head>
		<title>Wide-angle camera image</title>
		<link rel="stylesheet" type="text/css" href="layout.css">
		<meta http-equiv="REFRESH" content="900">
		<script src="imgBig.js"> </script>
	</head>

	<body align="center" onload=imgProc("wide")>
		<h3 align="center">Wide-angle Camera</h3>
		<img width="512" height="512" name="wideCamLastImg" alt="last wide-angle camera
image...">
		<br>
		<div id="filename"> </div>
	</body>
</html>
```

imgBig.js

```
// This script checks the status XML file of a camera and refreshes the image
 // if the filename has changed.

 var xmlStatLocTele = "tele/telecamStatus.xml";      // telescope camera & optics status xml
(from back-end driver)
 var xmlStatLocWide = "wide/wacamStatus.xml";        // wide-angle camera & optics status xml
(from back-end driver)

 ////////////////////////////////////////////////////////////////////////
 var xmlFile;
 var xmlHWDoc;

 var imgFile;
 var imgCall;

 var imgName;
 var lastImgName;

 var imgId
 var camera


 function imgProc(camSelect)
 {
	camera = camSelect
	if (camera == "tele")
	{
		xmlFile = xmlStatLocTele;
		imgId   = "teleCamLastImg";
	}
	else if (camera == "wide")
```

```
       {
              xmlFile = xmlStatLocWide;
              imgId   = "wideCamLastImg";
       }

       imgLoadXML();
}


function imgLoadXML()
{
   // code for IE
   if (window.ActiveXObject)
   {
        xmlHWDoc = new ActiveXObject("Microsoft.XMLDOM");
        xmlHWDoc.async = false;
        xmlHWDoc.load(xmlFile);
        checkImg();
   }
   // code for Mozilla, Firefox, Opera, etc.
   else if (document.implementation && document.implementation.createDocument)
   {
        xmlHWDoc = document.implementation.createDocument("", "", null);
        xmlHWDoc.load(xmlFile);
        xmlHWDoc.onload = checkImg;
   }
   else
   {
        alert('Your browser cannot handle this script');
   }

   setTimeout("imgLoadXML()", 4000)
}


function checkImg()
{

    var imgCall;

    if (camera == "tele") {  imgFile = "tele/"; }
    else if (camera == "wide") {  imgFile = "wide/"; }

    imgCall = xmlHWDoc.getElementsByTagName("camStatus")[0];

    imgName = imgCall.getElementsByTagName("lastPicName")[0].childNodes[0].nodeValue;

    document.getElementById("filename").innerHTML = imgName;

    if (!(imgName == lastImgName))
    {
       imgFile = imgFile +
imgCall.getElementsByTagName("lastPicPath")[0].childNodes[0].nodeValue;
        imgFile = imgFile + imgName;
       tmp = new Date();
       tmp = "?"+tmp.getTime()
       document.images[imgId].src = imgFile + tmp;
    }

    lastImgName = imgName

}
```

mount.js

```
var xmlStatLoc = "wide/mountStatus.xml";
 var xmlPrefsLoc = "mntUserPrefs.xml";
 var xmlStatDoc;
 var xmlPrefsDoc;
 var firstTime = true;
 var mountStatus;
```

```
 var lastMountStatus;

 function loadXML()
 {
   // code for IE
   if (window.ActiveXObject)
   {
         xmlStatDoc = new ActiveXObject("Microsoft.XMLDOM");
         xmlStatDoc.async = false;
         xmlStatDoc.load(xmlStatLoc);
         getData();
   }
   // code for Mozilla, Firefox, Opera, etc.
   else if (document.implementation && document.implementation.createDocument)
   {
         xmlStatDoc = document.implementation.createDocument("", "", null);
         xmlStatDoc.load(xmlStatLoc);
         xmlStatDoc.onload = getData;
   }
   else
   {
         alert('Your browser cannot handle this script');
   }
 }


////////////////////////////////////////////////////////////////////////////////////////
// Update current status of the mount
 function getData()
 {
   mountStatus = xmlStatDoc.getElementsByTagName("mountStatus")[0];
   var posStatus =
xmlStatDoc.getElementsByTagName("mountStatus")[0].getElementsByTagName("positionState")[0];

   // Display any error messages on the screen: Any text in the <error> field of status XML field
is displayed on top of page
   // The text doesn't show up if the XML field value is "FALSE"
   // (feature is disabled because error field is not being written by backenddriver!)
  // var errorVal = xmlStatDoc.getElementsByTagName("error")[0].childNodes[0].nodeValue;
  // if (errorVal != "false")
  // {
  //   document.getElementById("errorDisplay").innerHTML = '<div class="error">Error: ' +
errorVal + '</div><br>';
  // }
  // else
  // {
  //   document.getElementById("errorDisplay").innerHTML = "";
  // }

   // Extract the status of the mount hardware:
   var connectedStatus =
mountStatus.getElementsByTagName("connectedState")[0].childNodes[0].nodeValue;
   var slewingStatus =
mountStatus.getElementsByTagName("slewingState")[0].childNodes[0].nodeValue;
   var homingStatus = mountStatus.getElementsByTagName("homingState")[0].childNodes[0].nodeValue;

   // mountStatus tells user whether mount is Connected, Disconnected, Slewing, or Homing
   if          (connectedStatus == "FALSE")  {mountStatus = "Disconnected";}
   else if     (homingStatus == "TRUE")      {mountStatus = "Homing";}
   else if     (slewingStatus == "TRUE")     {mountStatus = "Slewing";}
   else if     (connectedStatus == "TRUE")   {mountStatus = "Connected";}
   else                                      {mountStatus = "Disconnected";}


   // Write the staus of the mount to the web page for the user to see
   document.getElementById("statusDisplay").innerHTML = mountStatus;
   document.getElementById("mountUpdated").innerHTML =
xmlStatDoc.getElementsByTagName("updated")[0].childNodes[0].nodeValue;

   // Extract the current position of the mount:
   if (connectedStatus == "TRUE")
```

```
   {
           document.getElementById("aziDegDisplay").innerHTML =
posStatus.getElementsByTagName("aziDegState")[0].childNodes[0].nodeValue;
           document.getElementById("aziMinDisplay").innerHTML =
posStatus.getElementsByTagName("aziMinState")[0].childNodes[0].nodeValue;
           document.getElementById("aziSecDisplay").innerHTML =
posStatus.getElementsByTagName("aziSecState")[0].childNodes[0].nodeValue;
           document.getElementById("altDegDisplay").innerHTML =
posStatus.getElementsByTagName("altDegState")[0].childNodes[0].nodeValue;
           document.getElementById("altMinDisplay").innerHTML =
posStatus.getElementsByTagName("altMinState")[0].childNodes[0].nodeValue;
           document.getElementById("altSecDisplay").innerHTML =
posStatus.getElementsByTagName("altSecState")[0].childNodes[0].nodeValue;
   }
   else if (connectedStatus == "FALSE")
   {
           document.getElementById("aziDegDisplay").innerHTML = "n/a";
           document.getElementById("aziMinDisplay").innerHTML = "n/a";
           document.getElementById("aziSecDisplay").innerHTML = "n/a";
           document.getElementById("altDegDisplay").innerHTML = "n/a";
           document.getElementById("altMinDisplay").innerHTML = "n/a";
           document.getElementById("altSecDisplay").innerHTML = "n/a";
   }



   // Check whether the mount status has changed, and reload the controls in case it has
   if ((firstTime == true) || (mountStatus != lastMountStatus))
   {
               loadUserPrefs();
               firstTime = false;
   }

   lastMountStatus = mountStatus;
   setTimeout("loadXML()",1000);

 }


////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////
function loadUserPrefs()
{
       if (window.ActiveXObject)
       {
                xmlPrefsDoc = new ActiveXObject("Microsoft.XMLDOM");
                xmlPrefsDoc.async = false;
                xmlPrefsDoc.load(xmlPrefsLoc);
                controlField();
       }
       else if (document.implementation && document.implementation.createDocument)
       {
                xmlPrefsDoc = document.implementation.createDocument("", "", null);
                xmlPrefsDoc.load(xmlPrefsLoc);
                xmlPrefsDoc.onload = controlField;
       }
}


////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////
function controlField()
{
       var mountPosPrefs = xmlPrefsDoc.getElementsByTagName("mountPrefs")[0];

       if (mountStatus == 'Connected')
       {
               // does not allow changes while an observation is in progress
               document.getElementById("setAziDegDisp").innerHTML = '<input type="text"
name="mntAziDeg" size="5">';
```

```
                document.getElementById("setAziMinDisp").innerHTML = '<input type="text"
name="mntAziMin" size="5">';
                document.getElementById("setAziSecDisp").innerHTML = '<input type="text"
name="mntAziSec" size="5">';
                document.getElementById("setAltDegDisp").innerHTML = '<input type="text"
name="mntAltDeg" size="5">';
                document.getElementById("setAltMinDisp").innerHTML = '<input type="text"
name="mntAltMin" size="5">';
                document.getElementById("setAltSecDisp").innerHTML = '<input type="text"
name="mntAltSec" size="5">';

                document.getElementById("slewButton").disabled=false;
                document.getElementById("abortButton").disabled=true;
                document.getElementById("connectButton").disabled=true;
                document.getElementById("homeButton").disabled=false;
                document.getElementById("disconnectButton").disabled=false;
        }
        else if (mountStatus == 'Disconnected')
        {
                document.getElementById("setAziDegDisp").innerHTML = "n/a";
                document.getElementById("setAziMinDisp").innerHTML = "n/a";
                document.getElementById("setAziSecDisp").innerHTML = "n/a";
                document.getElementById("setAltDegDisp").innerHTML = "n/a";
                document.getElementById("setAltMinDisp").innerHTML = "n/a";
                document.getElementById("setAltSecDisp").innerHTML = "n/a";

                document.getElementById("slewButton").disabled=true;
                document.getElementById("homeButton").disabled=true;
                document.getElementById("disconnectButton").disabled=true;

                document.getElementById("abortButton").disabled=true;
                document.getElementById("connectButton").disabled=false;

        }
        else if (mountStatus == 'Homing')
        {
                document.getElementById("setAziDegDisp").innerHTML = "n/a";
                document.getElementById("setAziMinDisp").innerHTML = "n/a";
                document.getElementById("setAziSecDisp").innerHTML = "n/a";
                document.getElementById("setAltDegDisp").innerHTML = "n/a";
                document.getElementById("setAltMinDisp").innerHTML = "n/a";
                document.getElementById("setAltSecDisp").innerHTML = "n/a";

                document.getElementById("slewButton").disabled=true;
                document.getElementById("homeButton").disabled=true;
                document.getElementById("disconnectButton").disabled=true;

                document.getElementById("abortButton").disabled=false;
                document.getElementById("connectButton").disabled=true;

        }
        else
        {
                // form allows changes when no observation in progress
                document.getElementById("setAziDegDisp").innerHTML =
mountPosPrefs.getElementsByTagName("mntAziDeg")[0].childNodes[0].nodeValue;
                document.getElementById("setAziMinDisp").innerHTML =
mountPosPrefs.getElementsByTagName("mntAziMin")[0].childNodes[0].nodeValue;
                document.getElementById("setAziSecDisp").innerHTML =
mountPosPrefs.getElementsByTagName("mntAziSec")[0].childNodes[0].nodeValue;
                document.getElementById("setAltDegDisp").innerHTML =
mountPosPrefs.getElementsByTagName("mntAltDeg")[0].childNodes[0].nodeValue;
                document.getElementById("setAltMinDisp").innerHTML =
mountPosPrefs.getElementsByTagName("mntAltMin")[0].childNodes[0].nodeValue;
                document.getElementById("setAltSecDisp").innerHTML =
mountPosPrefs.getElementsByTagName("mntAltSec")[0].childNodes[0].nodeValue;

                document.getElementById("slewButton").disabled=true;
                document.getElementById("homeButton").disabled=true;
                document.getElementById("disconnectButton").disabled=true;
```

```
                document.getElementById("abortButton").disabled=false;
                document.getElementById("connectButton").disabled=true;
        }
}


////////////////////////////////////////////////////////////////////////////////////////
// This function validates the user input in the position boxes to make sure it is in the right
format
function validate_form( )
{
    var valid = true;
    var checkme;

    checkme = parseInt(document.mountSlewForm.mntAziDeg.value,10);
    if (isNaN(checkme) || (0 > checkme) || (checkme > 359))
    {
        alert ( "Invalid value entered for Azimuth degrees" );
        valid = false;
    }

    checkme = parseInt(document.mountSlewForm.mntAziMin.value,10);
    if (isNaN(checkme) || (0 > checkme) || (checkme > 59))
    {
        alert ( "Invalid value entered for Azimuth minutes" );
        valid = false;
    }

    checkme = parseInt(document.mountSlewForm.mntAziSec.value,10);
    if (isNaN(checkme) || (0 > checkme) || (checkme > 59.9))
    {
        alert ( "Invalid value entered for Azimuth seconds" );
        valid = false;
    }

    checkme = parseInt(document.mountSlewForm.mntAltDeg.value,10);
    if (isNaN(checkme) || (0 > checkme) || (checkme > 90))
    {
        alert ( "Invalid value entered for Altitude degrees" );
        valid = false;
    }

    checkme = parseInt(document.mountSlewForm.mntAltMin.value,10);
    if (isNaN(checkme) || (0 > checkme) || (checkme > 59))
    {
        alert ( "Invalid value entered for Altitude minutes" );
        valid = false;
    }

    checkme = parseInt(document.mountSlewForm.mntAltSec.value,10);
    if (isNaN(checkme) || (0 > checkme) || (checkme > 59.9))
    {
        alert ( "Invalid value entered for Altitue seconds" );
        valid = false;
    }

    return valid;
}
```

observation.js

```
// Locations of XML documents:

 var xmlStatLocTele = "tele/telecamStatus.xml";      // telescope camera & optics status xml
(from back-end driver)
 var xmlStatLocWide = "wide/wacamStatus.xml";        // wide-angle camera & optics status xml
(from back-end driver)
 var xmlPrefsLoc = "obsUserPrefs.xml";               // user preferences XML (from Python script)

//var teleImgLoc = "teleCamLastImg.jpg";             // path and filename of last telescope image
```

```
// var wideImgLoc = "wideCamLastImg.jpg";           // path and filename of last wide-angle
image


////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////
 // NOTE for developers: The back-end driver writes all information (except field names)
 // to the XML file in Capital letters. That is, a value of "TRUE" will always be in caps, etc.
 // Javascript is case sensitive and will respond differently to "true" and "TRUE"
 // For convenience, we treat all TRUE/FALSE values written by the Javascript in caps as well.
 ////////////////////////////////////////////////////////////////////////////////////////////

 // define global variables
 var xmlHWDocTele;
 var xmlHWDocWide;
 var xmlUserDoc;

 var lastTeleImgName;
 var lastWideImgName;

 var firstTime = 1;
 var teleObsProgStatus;
 var waObsProgStatus;
 var lastTeleObsProgStatus;
 var lastWaObsProgStatus;

 var teleCamConnStat;
 var waCamConnStat;
 var teleOptConnStat;
 var waOptConnStat;

 var lastTeleCamConnStat;
 var lastWaCamConnStat;
 var lastTeleOptConnStat;
 var lastWaOptConnStat;


 function loadXML()
 {
   // code for IE
   if (window.ActiveXObject)
   {
         xmlHWDocTele = new ActiveXObject("Microsoft.XMLDOM");
         xmlHWDocTele.async = false;
         xmlHWDocTele.load(xmlStatLocTele);

         loadXML2();
   }
   // code for Mozilla, Firefox, Opera, etc.
   else if (document.implementation && document.implementation.createDocument)
   {
         xmlHWDocTele = document.implementation.createDocument("", "", null);
         xmlHWDocTele.load(xmlStatLocTele);
         xmlHWDocTele.onload = loadXML2;
   }
   else
   {
         alert('Your browser cannot handle this script');
   }
 }

 function loadXML2()
 {
    // code for IE
    if (window.ActiveXObject)
    {
         xmlHWDocWide = new ActiveXObject("Microsoft.XMLDOM");
         xmlHWDocWide.async = false;
         xmlHWDocWide.load(xmlStatLocWide);

         updateStatus();
```

```
    }
    // code for Mozilla, Firefox, Opera, etc.
    else if (document.implementation && document.implementation.createDocument)
    {
         xmlHWDocWide = document.implementation.createDocument("", "", null);
         xmlHWDocWide.load(xmlStatLocWide);
         xmlHWDocWide.onload = updateStatus;
   }

 }


////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////
// Function updates an image
// it takes in an image id and filename of the image source
 function updateImg(idname, filename)
 {
              tmp = new Date();
              tmp = "?"+tmp.getTime()
              document.images[idname].src = filename + tmp;
 }

////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////
// Update current status of hardware
 function updateStatus()
 {
   /// Define paths within the XML file to observation and hardware status ///
   var obsTeleStatus =
xmlHWDocTele.getElementsByTagName("camStatus")[0].getElementsByTagName("obs")[0];
   var obsWideStatus =
xmlHWDocWide.getElementsByTagName("camStatus")[0].getElementsByTagName("obs")[0];

   var hwTeleStatus =
xmlHWDocTele.getElementsByTagName("camStatus")[0].getElementsByTagName("hardware")[0];
   var hwWideStatus =
xmlHWDocWide.getElementsByTagName("camStatus")[0].getElementsByTagName("hardware")[0];

   teleCamConnStat =
hwTeleStatus.getElementsByTagName("camera")[0].getElementsByTagName("connectedState")[0].childNod
es[0].nodeValue;
   waCamConnStat =
hwWideStatus.getElementsByTagName("camera")[0].getElementsByTagName("connectedState")[0].childNod
es[0].nodeValue;
   teleOptConnStat =
hwTeleStatus.getElementsByTagName("optics")[0].getElementsByTagName("connectedState")[0].childNod
es[0].nodeValue;
   waOptConnStat =
hwWideStatus.getElementsByTagName("optics")[0].getElementsByTagName("connectedState")[0].childNod
es[0].nodeValue;

   teleObsProgStatus =
obsTeleStatus.getElementsByTagName("inProgress")[0].childNodes[0].nodeValue;
   waObsProgStatus = obsWideStatus.getElementsByTagName("inProgress")[0].childNodes[0].nodeValue;



       // Display any error messages on the screen:
       // The contents of XML field <error> display on the top of the page. If the field value
is "FALSE", then nothing displays
       // This feature is disabled because backend driver does not write this XML field.
//     var errorVal =
xmlHWDocTele.getElementsByTagName("camStatus")[0].getElementsByTagName("error")[0].childNodes[0].
nodeValue;
//     if (errorVal != "FALSE")
//     {
//             document.getElementById("teleErrorDisplay").innerHTML = '<div
class="error">Telescope error: ' + errorVal + '</div><br>';
//     }
```

```
//      else
//      {
//              document.getElementById("teleErrorDisplay").innerHTML = "";
//      }
//      var errorVal =
xmlHWDocWide.getElementsByTagName("camStatus")[0].getElementsByTagName("error")[0].childNodes[0].
nodeValue;
//      if (errorVal != "FALSE")
//      {
//              document.getElementById("waErrorDisplay").innerHTML = '<div class="error">Wide-
angle error: ' + errorVal + '</div><br>';
//      }
//      else
//      {
//              document.getElementById("waErrorDisplay").innerHTML = "";
//      }




   /// Write the status of the telescope camera: ///
   if (teleCamConnStat == "TRUE")
   {
        document.getElementById("teleCamStatus").innerHTML = "Connected";

        /// Don't display the connect/disconnect buttons if an observation is in progress:///
        if (teleObsProgStatus == "TRUE")
        {
                document.getElementById("teleCamConn").innerHTML = " ";
        }
        else
        {
                document.getElementById("teleCamConn").innerHTML = '<input type="hidden"
name="camConnect" value="FALSE"><input type="submit" name="submit" value="Disconnect Camera"
class="button">';
        }

        document.getElementById("teleCamCCDtemp").innerHTML =
hwTeleStatus.getElementsByTagName("camera")[0].getElementsByTagName("ccdTempState")[0].childNodes
[0].nodeValue;
   }
   else
   {
        document.getElementById("teleCamStatus").innerHTML = "Disconnected";
        document.getElementById("teleCamConn").innerHTML = '<input type="hidden"
name="camConnect" value="TRUE"><input type="submit" name="submit" value="Connect Camera"
class="button">';
        document.getElementById("teleCamCCDtemp").innerHTML = "n/a";
   }


   /// Write the status of the wide-angle camera: ///
   if (waCamConnStat == "TRUE")
   {
        document.getElementById("waCamStatus").innerHTML = "Connected";

        if (waObsProgStatus == "TRUE")
        {
                document.getElementById("waCamConn").innerHTML = " ";
        }
        else
        {
                document.getElementById("waCamConn").innerHTML = '<input type="hidden"
name="camConnect" value="FALSE"><input type="submit" name="submit" value="Disconnect Camera"
class="button">';
        }

        document.getElementById("waCamCCDtemp").innerHTML =
hwWideStatus.getElementsByTagName("camera")[0].getElementsByTagName("ccdTempState")[0].childNodes
[0].nodeValue;
```

```
   }
   else
   {
        document.getElementById("waCamStatus").innerHTML = "Disconnected";
        document.getElementById("waCamConn").innerHTML = '<input type="hidden" name="camConnect"
value="TRUE"><input type="submit" name="submit" value="Connect Camera" class="button">';
        document.getElementById("waCamCCDtemp").innerHTML = "n/a";
   }


   /// Write the status of the telescope optics: ///
   if (teleOptConnStat == "TRUE")
   {
        document.getElementById("teleOptStatus").innerHTML = "Connected";

        if (teleObsProgStatus == "TRUE")
        {
               document.getElementById("teleOptConn").innerHTML = " ";
        }
        else
        {
               document.getElementById("teleOptConn").innerHTML = '<input type="hidden"
name="optConnect" value="FALSE"><input type="submit" name="submit" value="Disconnect Optics"
class="button">';
        }

        document.getElementById("teleOptShutter").innerHTML =
hwTeleStatus.getElementsByTagName("optics")[0].getElementsByTagName("shutterState")[0].childNodes
[0].nodeValue;
        document.getElementById("teleOptFilter").innerHTML =
hwTeleStatus.getElementsByTagName("optics")[0].getElementsByTagName("filterState")[0].childNodes[
0].nodeValue;
   }
   else
   {
        document.getElementById("teleOptStatus").innerHTML = "Disconnected";
        document.getElementById("teleOptConn").innerHTML = '<input type="hidden"
name="optConnect" value="TRUE"><input type="submit" name="submit" value="Connect Optics"
class="button">';
        document.getElementById("teleOptShutter").innerHTML = "n/a";
        document.getElementById("teleOptFilter").innerHTML = "n/a";
   }



   /// Write the status of the wide-angle optics: ///
   if (waOptConnStat == "TRUE")
   {
        document.getElementById("waOptStatus").innerHTML = "Connected";

        if (waObsProgStatus == "TRUE")
        {
               document.getElementById("waOptConn").innerHTML = " ";
        }
        else
        {
               document.getElementById("waOptConn").innerHTML = '<input type="hidden"
name="optConnect" value="FALSE"><input type="submit" name="submit" value="Disconnect Optics"
class="button">';
        }

        document.getElementById("waOptShutter").innerHTML =
hwWideStatus.getElementsByTagName("optics")[0].getElementsByTagName("shutterState")[0].childNodes
[0].nodeValue;
        document.getElementById("waOptFilter").innerHTML =
hwWideStatus.getElementsByTagName("optics")[0].getElementsByTagName("filterState")[0].childNodes[
0].nodeValue;
   }
   else
   {
        document.getElementById("waOptStatus").innerHTML = "Disconnected";
```

```
        document.getElementById("waOptConn").innerHTML = '<input type="hidden" name="optConnect"
value="TRUE"><input type="submit" name="submit" value="Connect Optics" class="button">';
        document.getElementById("waOptShutter").innerHTML = "n/a";
        document.getElementById("waOptFilter").innerHTML = "n/a";
    }


    /// If necessary, update the images ///
    var imgFolder;
    var imgName;

    imgFolder = "tele/" +
obsTeleStatus.getElementsByTagName("lastPicPath")[0].childNodes[0].nodeValue;
    imgName = imgFolder +
obsTeleStatus.getElementsByTagName("lastPicName")[0].childNodes[0].nodeValue;
    if (!(imgName == lastTeleImgName))
    {
        updateImg("teleCamLastImg",imgName);
        document.getElementById("teleImgInfo").innerHTML =
obsTeleStatus.getElementsByTagName("lastPicName")[0].childNodes[0].nodeValue;
    }
    lastTeleImgName = imgName;

    imgFolder = "wide/" +
obsWideStatus.getElementsByTagName("lastPicPath")[0].childNodes[0].nodeValue;
    imgName = imgFolder +
obsWideStatus.getElementsByTagName("lastPicName")[0].childNodes[0].nodeValue;
    if (!(imgName == lastWideImgName))
    {
        updateImg("waCamLastImg",imgName);
        document.getElementById("waImgInfo").innerHTML =
obsWideStatus.getElementsByTagName("lastPicName")[0].childNodes[0].nodeValue;
    }
    lastWideImgName = imgName;


    /// Display the last updated time on the page ///
    document.getElementById("teleCamUpdated").innerHTML =
xmlHWDocTele.getElementsByTagName("camStatus")[0].getElementsByTagName("updated")[0].childNodes[0
].nodeValue;
    document.getElementById("waCamUpdated").innerHTML =
xmlHWDocWide.getElementsByTagName("camStatus")[0].getElementsByTagName("updated")[0].childNodes[0
].nodeValue;


    // Enable stop button if observation in progress:
    if (teleObsProgStatus == "TRUE" || waObsProgStatus == "TRUE")
    {
        document.getElementById("obsStartOp").innerHTML = '<input type="hidden" name="obsStart"
value="FALSE">';
    }
    else
    {
        document.getElementById("obsStartOp").innerHTML = '<input type="hidden" name="obsStart"
value="TRUE">';
    }


    // if both camera and optics connected, specify sending to them in observation (this allows
the stop button to work even if there is a page error)
    if (teleCamConnStat == "TRUE" && teleOptConnStat == "TRUE")
    {
        document.getElementById("teleSendToField").innerHTML = '<input type="hidden"
name="teleSendTo" value="TRUE">'
    }
    else
    {
        document.getElementById("teleSendToField").innerHTML = '<input type="hidden"
name="teleSendTo" value="FALSE">'
    }
    if (waCamConnStat == "TRUE" && waOptConnStat == "TRUE")
```

```
   {
        document.getElementById("waSendToField").innerHTML = '<input type="hidden"
name="waSendTo" value="TRUE">'
   }
   else
   {
       document.getElementById("waSendToField").innerHTML = '<input type="hidden"
name="waSendTo" value="FALSE">'
   }


   ///// Determine if the user control panel must be refreshed /////
   // If this is the first time through, the observation status changed, or a piece of
equipment's conenction status changed, then update the control fields //
   if ((firstTime == 1) || (teleObsProgStatus != lastTeleObsProgStatus) || (waObsProgStatus !=
lastWaObsProgStatus) || (teleCamConnStat != lastTeleCamConnStat) || (waCamConnStat !=
lastWaCamConnStat) || (teleOptConnStat != lastTeleOptConnStat) || (waOptConnStat !=
lastWaOptConnStat))
   {
               loadUserPrefs();
               firstTime = 0;
   }


   // Save the current observation status for the next iteration //
   lastTeleObsProgStatus = teleObsProgStatus;
   lastWaObsProgStatus = waObsProgStatus;
   lastTeleCamConnStat = teleCamConnStat;
   lastWaCamConnStat = waCamConnStat;
   lastTeleOptConnStat = teleOptConnStat;
   lastWaOptConnStat = waOptConnStat;


   // Call the function again after 1 second //
   setTimeout("loadXML()",3500);

 }



////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////
function controlField()
{
       ///// do not allow changes while an observation is in progress /////

       // start reading xml data: //
       var xmlUserPrefs = xmlUserDoc.getElementsByTagName("userPrefs")[0];


       // Write the fields that contain the user's preferences: //

       if (teleObsProgStatus == "TRUE" || waObsProgStatus == "TRUE")
       {
               // First, check whether equipment is connected; if not, then don't give user the
option of entering prefs for that equipment
               if (teleCamConnStat == "TRUE" && teleOptConnStat == "TRUE" && teleObsProgStatus ==
"TRUE")
               {
                       document.getElementById("teleObsADCrateDisp").innerHTML =
xmlUserPrefs.getElementsByTagName("tele")[0].getElementsByTagName("camADCrate")[0].childNodes[0].
nodeValue;
                       document.getElementById("teleObsExpTimeDisp").innerHTML =
xmlUserPrefs.getElementsByTagName("tele")[0].getElementsByTagName("camExpTime")[0].childNodes[0].
nodeValue + ' sec';
                       document.getElementById("teleObsFilterDisp").innerHTML =
xmlUserPrefs.getElementsByTagName("tele")[0].getElementsByTagName("optFilter")[0].childNodes[0].n
odeValue;
                       document.getElementById("teleObsShutterDisp").innerHTML =
xmlUserPrefs.getElementsByTagName("tele")[0].getElementsByTagName("optShutter")[0].childNodes[0].
nodeValue;
```

```
		}
		else
		{
			document.getElementById("teleObsADCrateDisp").innerHTML = "n/a";
			document.getElementById("teleObsExpTimeDisp").innerHTML = "n/a";
			document.getElementById("teleObsFilterDisp").innerHTML = "n/a";
			document.getElementById("teleObsShutterDisp").innerHTML = "n/a";
		}


		if (waCamConnStat == "TRUE" && waOptConnStat == "TRUE" && waObsProgStatus ==
"TRUE")
		{
			document.getElementById("waObsADCrateDisp").innerHTML =
xmlUserPrefs.getElementsByTagName("wide")[0].getElementsByTagName("camADCrate")[0].childNodes[0].
nodeValue;
			document.getElementById("waObsExpTimeDisp").innerHTML =
xmlUserPrefs.getElementsByTagName("wide")[0].getElementsByTagName("camExpTime")[0].childNodes[0].
nodeValue + ' sec';
			document.getElementById("waObsFilterDisp").innerHTML =
xmlUserPrefs.getElementsByTagName("wide")[0].getElementsByTagName("optFilter")[0].childNodes[0].n
odeValue;
			document.getElementById("waObsShutterDisp").innerHTML =
xmlUserPrefs.getElementsByTagName("wide")[0].getElementsByTagName("optShutter")[0].childNodes[0].
nodeValue;
		}
		else
		{
			document.getElementById("waObsADCrateDisp").innerHTML = "n/a";
			document.getElementById("waObsExpTimeDisp").innerHTML = "n/a";
			document.getElementById("waObsFilterDisp").innerHTML = "n/a";
			document.getElementById("waObsShutterDisp").innerHTML = "n/a";
		}

		document.getElementById("obsStartTimeDisp").innerHTML =
xmlUserPrefs.getElementsByTagName("obs")[0].getElementsByTagName("obsStartTime")[0].childNodes[0]
.nodeValue;
		document.getElementById("obsStopTimeDisp").innerHTML =
xmlUserPrefs.getElementsByTagName("obs")[0].getElementsByTagName("obsStopTime")[0].childNodes[0].
nodeValue;

		// Allow user to press Stop Observation Button, but not Start Observation
		document.getElementById("obsStartButton").disabled = true;
		document.getElementById("obsStopButton").disabled = false;
	}
	else		///// form allows changes when no observation in progress /////
	{
		// Write observation fields for Telescope camera and optics
		if (teleCamConnStat == "TRUE" && teleOptConnStat == "TRUE")
		{
			document.getElementById("teleObsADCrateDisp").innerHTML = '<select
name="teleCamADCrate"> <option value="fast">Fast</option> <option value="slow">Slow</option>
</select>';
			document.getElementById("teleObsExpTimeDisp").innerHTML = '<input
type="text" name="teleCamExpTime" size="6">  sec';
			document.getElementById("teleObsFilterDisp").innerHTML = ' <input
type="checkbox" name="teleOptFilter" value="1"> 1  <input type="checkbox" name="teleOptFilter"
value="2"> 2  <input type="checkbox" name="teleOptFilter" value="3"> 3 <br> <input
type="checkbox" name="teleOptFilter" value="4"> 4  <input type="checkbox" name="teleOptFilter"
value="5"> 5  <input type="checkbox" name="teleOptFilter" value="6"> 6';
			document.getElementById("teleObsShutterDisp").innerHTML = '<input
type="radio" name="teleOptShutter" value="open" checked> Open <br> <input type="radio"
name="teleOptShutter" value="close"> Closed';
		}
		else
		{
			document.getElementById("teleObsADCrateDisp").innerHTML = "n/a";
			document.getElementById("teleObsExpTimeDisp").innerHTML = "n/a";
			document.getElementById("teleObsFilterDisp").innerHTML = "n/a";
			document.getElementById("teleObsShutterDisp").innerHTML = "n/a";
		}
```

```
		if (waCamConnStat == "TRUE" && waOptConnStat == "TRUE")
		{
			document.getElementById("waObsADCrateDisp").innerHTML = '<select
name="waCamADCrate"> <option value="fast">Fast</option> <option value="slow">Slow</option>
</select>';
			document.getElementById("waObsExpTimeDisp").innerHTML = '<input
type="text" name="waCamExpTime" size="6">  sec';
			document.getElementById("waObsFilterDisp").innerHTML = ' <input
type="checkbox" name="waOptFilter" value="1"> 1  <input type="checkbox" name="waOptFilter"
value="2"> 2  <input type="checkbox" name="waOptFilter" value="3"> 3 <br> <input type="checkbox"
name="waOptFilter" value="4"> 4  <input type="checkbox" name="waOptFilter" value="5"> 5  <input
type="checkbox" name="waOptFilter" value="6"> 6';
			document.getElementById("waObsShutterDisp").innerHTML = '<input
type="radio" name="waOptShutter" value="open" checked> Open <br> <input type="radio"
name="waOptShutter" value="close"> Closed';
		}
		else
		{
			document.getElementById("waObsADCrateDisp").innerHTML = "n/a";
			document.getElementById("waObsExpTimeDisp").innerHTML = "n/a";
			document.getElementById("waObsFilterDisp").innerHTML = "n/a";
			document.getElementById("waObsShutterDisp").innerHTML = "n/a";
		}

		document.getElementById("obsStartTimeDisp").innerHTML = '<input type="text"
name="obsStartTimeHH" size="4"> : <input type="text" name="obsStartTimeMM" size="4"> : <input
type="text" name="obsStartTimeSS" size="4">';
		document.getElementById("obsStopTimeDisp").innerHTML = '<input type="text"
name="obsStopTimeHH" size="4"> : <input type="text" name="obsStopTimeMM" size="4"> : <input
type="text" name="obsStopTimeSS" size="4">';

		// Allow user to press Start Observation Button, but not Stop Observation
		document.getElementById("obsStartButton").disabled = false;
		document.getElementById("obsStopButton").disabled = true;
	}
}



////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////
function loadUserPrefs()
{
	if (window.ActiveXObject)
	{
		 xmlUserDoc = new ActiveXObject("Microsoft.XMLDOM");
		 xmlUserDoc.async = false;
		 xmlUserDoc.load(xmlPrefsLoc);
		 controlField();
	}
	else if (document.implementation && document.implementation.createDocument)
	{
		 xmlUserDoc = document.implementation.createDocument("", "", null);
		 xmlUserDoc.load(xmlPrefsLoc);
		 xmlUserDoc.onload = controlField;
	}
}

////////////////////////////////////////////////////////////////////////////////////
// Function checks form user input before sending it to the server

function validate_form()
{
    var valid = true;
    var checkme;

    // check telescope settings if telescope equipment connected
    if (teleCamConnStat == "TRUE" && teleOptConnStat == "TRUE")
    {
```

```
        checkme = parseFloat(document.obsForm.teleCamExpTime.value);
        if (isNaN(checkme) || (0 > checkme))
        {
                alert ("Invalid value entered for telescope exposure time");
                valid = false;
        }

    }

    // check wide-angle settings if wide-angle equipment connected
    if (waCamConnStat == "TRUE" && waOptConnStat == "TRUE")
    {

        checkme = parseFloat(document.obsForm.waCamExpTime.value,10);
        if (isNaN(checkme) || (0 > checkme))
        {
                alert ("Invalid value entered for wide-angle exposure time");
                valid = false;
        }

    }

    return valid;
}
```

Appendix E: Website XML Files

mntUserPrefs.xml

```
<mountPrefs>
	<mntAziDeg>202</mntAziDeg>
	<mntAziMin>0</mntAziMin>
	<mntAziSec>0</mntAziSec>
	<mntAltDeg>76</mntAltDeg>
	<mntAltMin>0</mntAltMin>
	<mntAltSec>0</mntAltSec>
</mountPrefs>
```

obsUserPrefs.xml

```
<userPrefs>
	<tele>
		<camADCrate>slow</camADCrate>
		<camExpTime>15000.0</camExpTime>
		<optFilter>2</optFilter>
		<optShutter>open</optShutter>
	</tele>
	<wide>
		<camADCrate>slow</camADCrate>
		<camExpTime>15000.0</camExpTime>
		<optFilter>2</optFilter>
		<optShutter>open</optShutter>
	</wide>
	<obs>
		<obsStartTime>5:2:0</obsStartTime>
		<obsStopTime>6:0:0</obsStopTime>
	</obs>
</userPrefs>
```

mountStatus.xml

```
<mountStatus>
	<updated>INV</updated>
	<connectedState>INV</connectedState>
	<slewingState>INV</slewingState>
	<homingState>INV</homingState>
	<positionState>
		<aziDegState>INV</aziDegState>
		<aziMinState>INV</aziMinState>
		<aziSecState>INV</aziSecState>
		<altDegState>INV</altDegState>
		<altMinState>INV</altMinState>
		<altSecState>INV</altSecState>
	</positionState>
</mountStatus>
```

wacamStatus.xml

```
<camStatus>
	<updated>?</updated>
	<obs>
		<inProgress>FALSE</inProgress>
```

```
            <lastPicName>default_image.png</lastPicName>
            <lastPicPath>./</lastPicPath>
      </obs>
      <hardware>
            <optics>
                  <connectedState>FALSE</connectedState>
                  <shutterState>OPEN</shutterState>
                  <filterState>1</filterState>
            </optics>
            <camera>
                  <connectedState>FALSE</connectedState>
                  <ccdTempState>?</ccdTempState>
            </camera>
      </hardware>
</camStatus>
```

telecamStatus.xml

```
<camStatus>
      <updated>?</updated>
      <obs>
            <inProgress>FALSE</inProgress>
            <lastPicName>TELE_2008-Mar-
      02_05_35_25_filt2_SLOW_15s.png</lastPicName>
            <lastPicPath>png/2008-Mar-02_05_01_24/</lastPicPath>
      </obs>
      <hardware>
            <optics>
                  <connectedState>TRUE</connectedState>
                  <shutterState>OPEN</shutterState>
                  <filterState>2</filterState>
            </optics>
            <camera>
                  <connectedState>TRUE</connectedState>
                  <ccdTempState>-40</ccdTempState>
            </camera>
      </hardware>
</camStatus>
```

Appendix F: Apache Server Configuration

httdp.conf

```
#
# This is the main Apache HTTP server configuration file.  It contains the
# configuration directives that give the server its instructions.
# See <URL:http://httpd.apache.org/docs/2.2> for detailed information.
# In particular, see
# <URL:http://httpd.apache.org/docs/2.2/mod/directives.html>
# for a discussion of each configuration directive.
#
# Do NOT simply read the instructions in here without understanding
# what they do.  They're here only as hints or reminders.  If you are unsure
# consult the online docs. You have been warned.
#
# Configuration and logfile names: If the filenames you specify for many
# of the server's control files begin with "/" (or "drive:/" for Win32), the
# server will use that explicit path.  If the filenames do *not* begin
# with "/", the value of ServerRoot is prepended -- so "logs/foo.log"
# with ServerRoot set to "C:/Program Files/Telescope Automation/Apache" will be interpreted by
the
# server as "C:/Program Files/Telescope Automation/Apache/logs/foo.log".
#
# NOTE: Where filenames are specified, you must use forward slashes
# instead of backslashes (e.g., "c:/apache" instead of "c:\apache").
# If a drive letter is omitted, the drive on which Apache.exe is located
# will be used by default.  It is recommended that you always supply
# an explicit drive letter in absolute paths to avoid confusion.

#
# ServerRoot: The top of the directory tree under which the server's
# configuration, error, and log files are kept.
#
# Do not add a slash at the end of the directory path.  If you point
# ServerRoot at a non-local disk, be sure to point the LockFile directive
# at a local disk.  If you wish to share the same ServerRoot for multiple
# httpd daemons, you will need to change at least LockFile and PidFile.
#
ServerRoot "C:/Program Files/Telescope Automation/Apache"

#
# Listen: Allows you to bind Apache to specific IP addresses and/or
# ports, instead of the default. See also the <VirtualHost>
# directive.
#
# Change this to Listen on specific IP addresses as shown below to
# prevent Apache from glomming onto all bound IP addresses.
#
#Listen 12.34.56.78:80
Listen 7999

#
# Dynamic Shared Object (DSO) Support
#
# To be able to use the functionality of a module which was built as a DSO you
# have to place corresponding `LoadModule' lines at this location so the
# directives contained in it are actually available _before_ they are used.
# Statically compiled modules (those listed by `httpd -l') do not need
# to be loaded here.
#
# Example:
# LoadModule foo_module modules/mod_foo.so
#
LoadModule actions_module modules/mod_actions.so
LoadModule alias_module modules/mod_alias.so
LoadModule asis_module modules/mod_asis.so
LoadModule auth_basic_module modules/mod_auth_basic.so
#LoadModule auth_digest_module modules/mod_auth_digest.so
#LoadModule authn_alias_module modules/mod_authn_alias.so
```

```
#LoadModule authn_anon_module modules/mod_authn_anon.so
#LoadModule authn_dbd_module modules/mod_authn_dbd.so
#LoadModule authn_dbm_module modules/mod_authn_dbm.so
LoadModule authn_default_module modules/mod_authn_default.so
LoadModule authn_file_module modules/mod_authn_file.so
#LoadModule authnz_ldap_module modules/mod_authnz_ldap.so
#LoadModule authz_dbm_module modules/mod_authz_dbm.so
LoadModule authz_default_module modules/mod_authz_default.so
LoadModule authz_groupfile_module modules/mod_authz_groupfile.so
LoadModule authz_host_module modules/mod_authz_host.so
#LoadModule authz_owner_module modules/mod_authz_owner.so
LoadModule authz_user_module modules/mod_authz_user.so
LoadModule autoindex_module modules/mod_autoindex.so
#LoadModule cache_module modules/mod_cache.so
#LoadModule cern_meta_module modules/mod_cern_meta.so
LoadModule cgi_module modules/mod_cgi.so
#LoadModule charset_lite_module modules/mod_charset_lite.so
#LoadModule dav_module modules/mod_dav.so
#LoadModule dav_fs_module modules/mod_dav_fs.so
#LoadModule dav_lock_module modules/mod_dav_lock.so
#LoadModule dbd_module modules/mod_dbd.so
#LoadModule deflate_module modules/mod_deflate.so
LoadModule dir_module modules/mod_dir.so
#LoadModule disk_cache_module modules/mod_disk_cache.so
#LoadModule dumpio_module modules/mod_dumpio.so
LoadModule env_module modules/mod_env.so
#LoadModule expires_module modules/mod_expires.so
#LoadModule ext_filter_module modules/mod_ext_filter.so
#LoadModule file_cache_module modules/mod_file_cache.so
#LoadModule filter_module modules/mod_filter.so
#LoadModule headers_module modules/mod_headers.so
#LoadModule ident_module modules/mod_ident.so
#LoadModule imagemap_module modules/mod_imagemap.so
LoadModule include_module modules/mod_include.so
#LoadModule info_module modules/mod_info.so
LoadModule isapi_module modules/mod_isapi.so
#LoadModule ldap_module modules/mod_ldap.so
#LoadModule logio_module modules/mod_logio.so
LoadModule log_config_module modules/mod_log_config.so
#LoadModule log_forensic_module modules/mod_log_forensic.so
#LoadModule mem_cache_module modules/mod_mem_cache.so
LoadModule mime_module modules/mod_mime.so
#LoadModule mime_magic_module modules/mod_mime_magic.so
LoadModule negotiation_module modules/mod_negotiation.so
#LoadModule proxy_module modules/mod_proxy.so
#LoadModule proxy_ajp_module modules/mod_proxy_ajp.so
#LoadModule proxy_balancer_module modules/mod_proxy_balancer.so
#LoadModule proxy_connect_module modules/mod_proxy_connect.so
#LoadModule proxy_ftp_module modules/mod_proxy_ftp.so
#LoadModule proxy_http_module modules/mod_proxy_http.so
#LoadModule rewrite_module modules/mod_rewrite.so
LoadModule setenvif_module modules/mod_setenvif.so
#LoadModule speling_module modules/mod_speling.so
LoadModule ssl_module modules/mod_ssl.so
#LoadModule status_module modules/mod_status.so
#LoadModule substitute_module modules/mod_substitute.so
#LoadModule unique_id_module modules/mod_unique_id.so
#LoadModule userdir_module modules/mod_userdir.so
#LoadModule usertrack_module modules/mod_usertrack.so
#LoadModule version_module modules/mod_version.so
#LoadModule vhost_alias_module modules/mod_vhost_alias.so

<IfModule !mpm_netware_module>
<IfModule !mpm_winnt_module>
#
# If you wish httpd to run as a different user or group, you must run
# httpd as root initially and it will switch.
#
# User/Group: The name (or #number) of the user/group to run httpd as.
# It is usually good practice to create a dedicated user and group for
# running httpd, as with most system services.
```

```
#
User daemon
Group daemon

</IfModule>
</IfModule>

# 'Main' server configuration
#
# The directives in this section set up the values used by the 'main'
# server, which responds to any requests that aren't handled by a
# <VirtualHost> definition.  These values also provide defaults for
# any <VirtualHost> containers you may define later in the file.
#
# All of these directives may appear inside <VirtualHost> containers,
# in which case these default settings will be overridden for the
# virtual host being defined.
#

#
# ServerAdmin: Your address, where problems with the server should be
# e-mailed.  This address appears on some server-generated pages, such
# as error documents.  e.g. admin@your-domain.com
#
ServerAdmin te08@wpi.edu

#
# ServerName gives the name and port that the server uses to identify itself.
# This can often be determined automatically, but we recommend you specify
# it explicitly to prevent problems during startup.
#
# If your host doesn't have a registered DNS name, enter its IP address here.
#
#ServerName WideFOV:7999

#
# DocumentRoot: The directory out of which you will serve your
# documents. By default, all requests are taken from this directory, but
# symbolic links and aliases may be used to point to other locations.
#
DocumentRoot "C:/Program Files/Telescope Automation/Apache/htdocs"

#
# Each directory to which Apache has access can be configured with respect
# to which services and features are allowed and/or disabled in that
# directory (and its subdirectories).
#
# First, we configure the "default" to be a very restrictive set of
# features.
#
<Directory />
    Options FollowSymLinks
    AllowOverride None
    Order deny,allow
    Deny from all
</Directory>

#
# Note that from this point forward you must specifically allow
# particular features to be enabled - so if something's not working as
# you might expect, make sure that you have specifically enabled it
# below.
#

#
# This should be changed to whatever you set DocumentRoot to.
#
<Directory "C:/Program Files/Telescope Automation/Apache/htdocs">
    #
    # Possible values for the Options directive are "None", "All",
    # or any combination of:
```

```
    #   Indexes Includes FollowSymLinks SymLinksifOwnerMatch ExecCGI MultiViews
    #
    # Note that "MultiViews" must be named *explicitly* --- "Options All"
    # doesn't give it to you.
    #
    # The Options directive is both complicated and important.  Please see
    # http://httpd.apache.org/docs/2.2/mod/core.html#options
    # for more information.
    #
    Options Indexes FollowSymLinks

    #
    # AllowOverride controls what directives may be placed in .htaccess files.
    # It can be "All", "None", or any combination of the keywords:
    #   Options FileInfo AuthConfig Limit
    #
    AllowOverride None

    #
    # Controls who can get stuff from this server.
    #
    Order allow,deny
    Allow from all

    AuthType Basic
    AuthName "SRI Access"
    AuthUserFile "C:/Program Files/Telescope Automation/Apache/conf/passwords"
    Require user sri
</Directory>

#
# DirectoryIndex: sets the file that Apache will serve if a directory
# is requested.
#
<IfModule dir_module>
    DirectoryIndex index.html
</IfModule>

#
# The following lines prevent .htaccess and .htpasswd files from being
# viewed by Web clients.
#
<FilesMatch "^\.ht">
    Order allow,deny
    Deny from all
    Satisfy All
</FilesMatch>

#
# ErrorLog: The location of the error log file.
# If you do not specify an ErrorLog directive within a <VirtualHost>
# container, error messages relating to that virtual host will be
# logged here.  If you *do* define an error logfile for a <VirtualHost>
# container, that host's errors will be logged there and not here.
#
ErrorLog "logs/error.log"

#
# LogLevel: Control the number of messages logged to the error_log.
# Possible values include: debug, info, notice, warn, error, crit,
# alert, emerg.
#
LogLevel warn

<IfModule log_config_module>
    #
    # The following directives define some format nicknames for use with
    # a CustomLog directive (see below).
    #
    LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined
    LogFormat "%h %l %u %t \"%r\" %>s %b" common
```

```
    <IfModule logio_module>
      # You need to enable mod_logio.c to use %I and %O
      LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" %I %O" combinedio
    </IfModule>

    #
    # The location and format of the access logfile (Common Logfile Format).
    # If you do not define any access logfiles within a <VirtualHost>
    # container, they will be logged here.  Contrariwise, if you *do*
    # define per-<VirtualHost> access logfiles, transactions will be
    # logged therein and *not* in this file.
    #
    CustomLog "logs/access.log" common

    #
    # If you prefer a logfile with access, agent, and referer information
    # (Combined Logfile Format) you can use the following directive.
    #
    #CustomLog "logs/access.log" combined
</IfModule>

<IfModule alias_module>
    #
    # Redirect: Allows you to tell clients about documents that used to
    # exist in your server's namespace, but do not anymore. The client
    # will make a new request for the document at its new location.
    # Example:
    # Redirect permanent /foo http://WideFOV/bar

    #
    # Alias: Maps web paths into filesystem paths and is used to
    # access content that does not live under the DocumentRoot.
    # Example:
    # Alias /webpath /full/filesystem/path
    #
    # If you include a trailing / on /webpath then the server will
    # require it to be present in the URL.  You will also likely
    # need to provide a <Directory> section to allow access to
    # the filesystem path.

    #
    # ScriptAlias: This controls which directories contain server scripts.
    # ScriptAliases are essentially the same as Aliases, except that
    # documents in the target directory are treated as applications and
    # run by the server when requested rather than as documents sent to the
    # client.  The same rules about trailing "/" apply to ScriptAlias
    # directives as to Alias.
    #
    ScriptAlias /cgi-bin/ "C:/Program Files/Telescope Automation/Apache/cgi-bin/"

</IfModule>

<IfModule cgid_module>
    #
    # ScriptSock: On threaded servers, designate the path to the UNIX
    # socket used to communicate with the CGI daemon of mod_cgid.
    #
    #Scriptsock logs/cgisock
</IfModule>

#
# "C:/Program Files/Telescope Automation/Apache/cgi-bin" should be changed to whatever your
ScriptAliased
# CGI directory exists, if you have that configured.
#
<Directory "C:/Program Files/Telescope Automation/Apache/cgi-bin">
    AllowOverride None
    Options None
    Order allow,deny
    Allow from all
```

```
</Directory>

#
# DefaultType: the default MIME type the server will use for a document
# if it cannot otherwise determine one, such as from filename extensions.
# If your server contains mostly text or HTML documents, "text/plain" is
# a good value.  If most of your content is binary, such as applications
# or images, you may want to use "application/octet-stream" instead to
# keep browsers from trying to display binary files as though they are
# text.
#
DefaultType text/plain

<IfModule mime_module>
    #
    # TypesConfig points to the file containing the list of mappings from
    # filename extension to MIME-type.
    #
    TypesConfig conf/mime.types

    #
    # AddType allows you to add to or override the MIME configuration
    # file specified in TypesConfig for specific file types.
    #
    #AddType application/x-gzip .tgz
    #
    # AddEncoding allows you to have certain browsers uncompress
    # information on the fly. Note: Not all browsers support this.
    #
    #AddEncoding x-compress .Z
    #AddEncoding x-gzip .gz .tgz
    #
    # If the AddEncoding directives above are commented-out, then you
    # probably should define those extensions to indicate media types:
    #
    AddType application/x-compress .Z
    AddType application/x-gzip .gz .tgz

    #
    # AddHandler allows you to map certain file extensions to "handlers":
    # actions unrelated to filetype. These can be either built into the server
    # or added with the Action directive (see below)
    #
    # To use CGI scripts outside of ScriptAliased directories:
    # (You will also need to add "ExecCGI" to the "Options" directive.)
    #
    #AddHandler cgi-script .cgi

    # For type maps (negotiated resources):
    #AddHandler type-map var

    #
    # Filters allow you to process content before it is sent to the client.
    #
    # To parse .shtml files for server-side includes (SSI):
    # (You will also need to add "Includes" to the "Options" directive.)
    #
    #AddType text/html .shtml
    #AddOutputFilter INCLUDES .shtml
</IfModule>

#
# The mod_mime_magic module allows the server to use various hints from the
# contents of the file itself to determine its type.  The MIMEMagicFile
# directive tells the module where the hint definitions are located.
#
#MIMEMagicFile conf/magic

#
# Customizable error responses come in three flavors:
# 1) plain text 2) local redirects 3) external redirects
```

```
#
# Some examples:
#ErrorDocument 500 "The server made a boo boo."
#ErrorDocument 404 /missing.html
#ErrorDocument 404 "/cgi-bin/missing_handler.pl"
#ErrorDocument 402 http://WideFOV/subscription_info.html
#

#
# EnableMMAP and EnableSendfile: On systems that support it,
# memory-mapping or the sendfile syscall is used to deliver
# files.  This usually improves server performance, but must
# be turned off when serving from networked-mounted
# filesystems or if support for these functions is otherwise
# broken on your system.
#
#EnableMMAP off
#EnableSendfile off

# Supplemental configuration
#
# The configuration files in the conf/extra/ directory can be
# included to add extra features or to modify the default configuration of
# the server, or you may simply copy their contents here and change as
# necessary.

# Server-pool management (MPM specific)
#Include conf/extra/httpd-mpm.conf

# Multi-language error messages
#Include conf/extra/httpd-multilang-errordoc.conf

# Fancy directory listings
#Include conf/extra/httpd-autoindex.conf

# Language settings
#Include conf/extra/httpd-languages.conf

# User home directories
#Include conf/extra/httpd-userdir.conf

# Real-time info on requests and configuration
#Include conf/extra/httpd-info.conf

# Virtual hosts
#Include conf/extra/httpd-vhosts.conf

# Local access to the Apache HTTP Server Manual
#Include conf/extra/httpd-manual.conf

# Distributed authoring and versioning (WebDAV)
#Include conf/extra/httpd-dav.conf

# Various default settings
#Include conf/extra/httpd-default.conf

# Secure (SSL/TLS) connections
Include conf/extra/httpd-ssl.conf
#
# Note: The following must must be present to support
#       starting without SSL on platforms with no /dev/random equivalent
#       but a statically compiled-in mod_ssl.
#
<IfModule ssl_module>
SSLRandomSeed startup builtin
SSLRandomSeed connect builtin
</IfModule>

Win32DisableAcceptEx
```

httpd-ssl.conf

```
#
# This is the Apache server configuration file providing SSL support.
# It contains the configuration directives to instruct the server how to
# serve pages over an https connection. For detailing information about these
# directives see <URL:http://httpd.apache.org/docs/2.2/mod/mod_ssl.html>
#
# Do NOT simply read the instructions in here without understanding
# what they do.  They're here only as hints or reminders.  If you are unsure
# consult the online docs. You have been warned.
#

#
# Pseudo Random Number Generator (PRNG):
# Configure one or more sources to seed the PRNG of the SSL library.
# The seed data should be of good random quality.
# WARNING! On some platforms /dev/random blocks if not enough entropy
# is available. This means you then cannot use the /dev/random device
# because it would lead to very long connection times (as long as
# it requires to make more entropy available). But usually those
# platforms additionally provide a /dev/urandom device which doesn't
# block. So, if available, use this one instead. Read the mod_ssl User
# Manual for more details.
#
#SSLRandomSeed startup file:/dev/random  512
#SSLRandomSeed startup file:/dev/urandom 512
#SSLRandomSeed connect file:/dev/random  512
#SSLRandomSeed connect file:/dev/urandom 512


#
# When we also provide SSL we have to listen to the
# standard HTTP port (see above) and to the HTTPS port
#
# Note: Configurations that use IPv6 but not IPv4-mapped addresses need two
#       Listen directives: "Listen [::]:8000" and "Listen 0.0.0.0:8000"
#
Listen 8000

##
##  SSL Global Context
##
##  All SSL configuration in this context applies both to
##  the main server and all SSL-enabled virtual hosts.
##

#
#   Some MIME-types for downloading Certificates and CRLs
#
AddType application/x-x509-ca-cert .crt
AddType application/x-pkcs7-crl    .crl

#   Pass Phrase Dialog:
#   Configure the pass phrase gathering process.
#   The filtering dialog program (`builtin' is a internal
#   terminal dialog) has to provide the pass phrase on stdout.
SSLPassPhraseDialog  builtin

#   Inter-Process Session Cache:
#   Configure the SSL Session Cache: First the mechanism
#   to use and second the expiring timeout (in seconds).
#SSLSessionCache         "dbm:C:/Program Files/Telescope Automation/Apache/logs/ssl_scache"
SSLSessionCache        "shmcb:C:/Program Files/Telescope
Automation/Apache/logs/ssl_scache(512000)"
SSLSessionCacheTimeout  300

#   Semaphore:
#   Configure the path to the mutual exclusion semaphore the
#   SSL engine uses internally for inter-process synchronization.
SSLMutex default
```

```
##
## SSL Virtual Host Context
##

<VirtualHost _default_:8000>

#   General setup for the virtual host
DocumentRoot "C:/Program Files/Telescope Automation/Apache/htdocs"
ServerName WideFOV:443
ServerAdmin te08@wpi.edu
ErrorLog "C:/Program Files/Telescope Automation/Apache/logs/error.log"
TransferLog "C:/Program Files/Telescope Automation/Apache/logs/access.log"

#   SSL Engine Switch:
#   Enable/Disable SSL for this virtual host.
SSLEngine on

#   SSL Cipher Suite:
#   List the ciphers that the client is permitted to negotiate.
#   See the mod_ssl documentation for a complete list.
SSLCipherSuite ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP:+eNULL

#   Server Certificate:
#   Point SSLCertificateFile at a PEM encoded certificate.  If
#   the certificate is encrypted, then you will be prompted for a
#   pass phrase.  Note that a kill -HUP will prompt again.  Keep
#   in mind that if you have both an RSA and a DSA certificate you
#   can configure both in parallel (to also allow the use of DSA
#   ciphers, etc.)
SSLCertificateFile "C:/Program Files/Telescope Automation/Apache/conf/haarp.cert"
#SSLCertificateFile "C:/Program Files/Telescope Automation/Apache/conf/server-dsa.crt"

#   Server Private Key:
#   If the key is not combined with the certificate, use this
#   directive to point at the key file.  Keep in mind that if
#   you've both a RSA and a DSA private key you can configure
#   both in parallel (to also allow the use of DSA ciphers, etc.)
SSLCertificateKeyFile "C:/Program Files/Telescope Automation/Apache/conf/haarp.key"
#SSLCertificateKeyFile "C:/Program Files/Telescope Automation/Apache/conf/server-dsa.key"

#   Server Certificate Chain:
#   Point SSLCertificateChainFile at a file containing the
#   concatenation of PEM encoded CA certificates which form the
#   certificate chain for the server certificate. Alternatively
#   the referenced file can be the same as SSLCertificateFile
#   when the CA certificates are directly appended to the server
#   certificate for convinience.
#SSLCertificateChainFile "C:/Program Files/Telescope Automation/Apache/conf/server-ca.crt"

#   Certificate Authority (CA):
#   Set the CA certificate verification path where to find CA
#   certificates for client authentication or alternatively one
#   huge file containing all of them (file must be PEM encoded)
#   Note: Inside SSLCACertificatePath you need hash symlinks
#         to point to the certificate files. Use the provided
#         Makefile to update the hash symlinks after changes.
#SSLCACertificatePath "C:/Program Files/Telescope Automation/Apache/conf/ssl.crt"
#SSLCACertificateFile "C:/Program Files/Telescope Automation/Apache/conf/ssl.crt/ca-bundle.crt"

#   Certificate Revocation Lists (CRL):
#   Set the CA revocation path where to find CA CRLs for client
#   authentication or alternatively one huge file containing all
#   of them (file must be PEM encoded)
#   Note: Inside SSLCARevocationPath you need hash symlinks
#         to point to the certificate files. Use the provided
#         Makefile to update the hash symlinks after changes.
#SSLCARevocationPath "C:/Program Files/Telescope Automation/Apache/conf/ssl.crl"
#SSLCARevocationFile "C:/Program Files/Telescope Automation/Apache/conf/ssl.crl/ca-bundle.crl"

#   Client Authentication (Type):
```

```
#   Client certificate verification type and depth.  Types are
#   none, optional, require and optional_no_ca.  Depth is a
#   number which specifies how deeply to verify the certificate
#   issuer chain before deciding the certificate is not valid.
#SSLVerifyClient require
#SSLVerifyDepth  10

#   Access Control:
#   With SSLRequire you can do per-directory access control based
#   on arbitrary complex boolean expressions containing server
#   variable checks and other lookup directives.  The syntax is a
#   mixture between C and Perl.  See the mod_ssl documentation
#   for more details.
#<Location />
#SSLRequire (    %{SSL_CIPHER} !~ m/^(EXP|NULL)/ \
#            and %{SSL_CLIENT_S_DN_O} eq "Snake Oil, Ltd." \
#            and %{SSL_CLIENT_S_DN_OU} in {"Staff", "CA", "Dev"} \
#            and %{TIME_WDAY} >= 1 and %{TIME_WDAY} <= 5 \
#            and %{TIME_HOUR} >= 8 and %{TIME_HOUR} <= 20       ) \
#           or %{REMOTE_ADDR} =~ m/^192\.76\.162\.[0-9]+$/
#</Location>

#   SSL Engine Options:
#   Set various options for the SSL engine.
#   o FakeBasicAuth:
#     Translate the client X.509 into a Basic Authorisation.  This means that
#     the standard Auth/DBMAuth methods can be used for access control.  The
#     user name is the `one line' version of the client's X.509 certificate.
#     Note that no password is obtained from the user. Every entry in the user
#     file needs this password: `xxj31ZMTZzkVA'.
#   o ExportCertData:
#     This exports two additional environment variables: SSL_CLIENT_CERT and
#     SSL_SERVER_CERT. These contain the PEM-encoded certificates of the
#     server (always existing) and the client (only existing when client
#     authentication is used). This can be used to import the certificates
#     into CGI scripts.
#   o StdEnvVars:
#     This exports the standard SSL/TLS related `SSL_*' environment variables.
#     Per default this exportation is switched off for performance reasons,
#     because the extraction step is an expensive operation and is usually
#     useless for serving static content. So one usually enables the
#     exportation for CGI and SSI requests only.
#   o StrictRequire:
#     This denies access when "SSLRequireSSL" or "SSLRequire" applied even
#     under a "Satisfy any" situation, i.e. when it applies access is denied
#     and no other module can change it.
#   o OptRenegotiate:
#     This enables optimized SSL connection renegotiation handling when SSL
#     directives are used in per-directory context.
#SSLOptions +FakeBasicAuth +ExportCertData +StrictRequire
<FilesMatch "\.(cgi|shtml|phtml|php)$">
    SSLOptions +StdEnvVars
</FilesMatch>
<Directory "C:/Program Files/Telescope Automation/Apache/cgi-bin">
    SSLOptions +StdEnvVars
</Directory>

#   SSL Protocol Adjustments:
#   The safe and default but still SSL/TLS standard compliant shutdown
#   approach is that mod_ssl sends the close notify alert but doesn't wait for
#   the close notify alert from client. When you need a different shutdown
#   approach you can use one of the following variables:
#   o ssl-unclean-shutdown:
#     This forces an unclean shutdown when the connection is closed, i.e. no
#     SSL close notify alert is send or allowed to received.  This violates
#     the SSL/TLS standard but is needed for some brain-dead browsers. Use
#     this when you receive I/O errors because of the standard approach where
#     mod_ssl sends the close notify alert.
#   o ssl-accurate-shutdown:
#     This forces an accurate shutdown when the connection is closed, i.e. a
#     SSL close notify alert is send and mod_ssl waits for the close notify
```

```
#     alert of the client. This is 100% SSL/TLS standard compliant, but in
#     practice often causes hanging connections with brain-dead browsers. Use
#     this only for browsers where you know that their SSL implementation
#     works correctly.
#   Notice: Most problems of broken clients are also related to the HTTP
#   keep-alive facility, so you usually additionally want to disable
#   keep-alive for those clients, too. Use variable "nokeepalive" for this.
#   Similarly, one has to force some clients to use HTTP/1.0 to workaround
#   their broken HTTP/1.1 implementation. Use variables "downgrade-1.0" and
#   "force-response-1.0" for this.
BrowserMatch ".*MSIE.*" \
         nokeepalive ssl-unclean-shutdown \
         downgrade-1.0 force-response-1.0

#   Per-Server Logging:
#   The home of a custom SSL log file. Use this when you want a
#   compact non-error SSL logfile on a virtual host basis.
CustomLog "C:/Program Files/Telescope Automation/Apache/logs/ssl_request.log" \
          "%t %h %{SSL_PROTOCOL}x %{SSL_CIPHER}x \"%r\" %b"

</VirtualHost>
```

Appendix G: CGI Script Python Code

control.py

```
#!C:/Python/python.exe
# this MUST point to the Python executable!!!

# import libraries:
import cgi, re, sys
from socket import *
import logging


#  ***** These settings should be set by the user for the current setup *******

# Location of XML files that store user preferences:
obsUserXML = 'C:\Program Files\Telescope Automation\Apache\htdocs\obsUserPrefs.xml'       # XML
for observation user prefs
mountUserXML = 'C:\Program Files\Telescope Automation\Apache\htdocs\mntUserPrefs.xml'     # XML
for mount user prefs

#IP addresses of computers connected to cameras and mount
teleCamIP = '***.***.**.**'
wideCamIP = '***.***.**.**'
mountIP = '***.***.**.**'

# web address of the observation control home page (include trailing /)
pageAddress = 'https://***.***.**.**:8000/'

#teleCamIP = '127.0.0.1'
#wideCamIP = '128.18.144.218'
#mountIP = '127.0.0.1'

# ************************** End of user settings **************************


###############################################################################
# This function sends a string over TCP/IP to the given IP address over Port 5000

def sendString(IPaddress, input) :

        HOST = IPaddress
        PORT = 5000                                     #data is sent to back-end driver over port
5000
        data = ''                                       #stores the confirmation code that back-end
driver sends back during communication
        s = socket(AF_INET, SOCK_STREAM)

        try:
                s.connect((HOST, PORT))
                s.settimeout(5)
        except  timeout:
                error = 'Connect timeout error! <br> Unable to reach address: ' + HOST
                errorPrint(error)
        except:
                error = 'Connect error! <br> Unable to reach address: ' + HOST
                errorPrint(error)

        try:
                s.send(input)                           # send the string
                s.settimeout(5)

                while (s.recv(1) != "") :               # wait until a null character is received
before continuing
                        pass                            # otherwise the connection will close and
back-end driver won't receive all commands

                s.close()                                       #close the connection
```

```
        except  timeout:
                error = 'Send timeout error! <br> Unable to reach address: ' + HOST
                errorPrint(error)
        except:
                error = 'Send error! <br> Unable to reach address: ' + HOST
                errorPrint(error)


################################################################################
# This handles errors. The 'try' statements throughout the program attempt to
# perform risky actions. If one doesn't work, the browser will simply print
# what the error, rather than returning "Internal server Error."


def errorPrint(errorCode):
        print 'The following error occurred: ' + errorCode + '<br> </HTML>'
        sys.exit()



################################################################################

###### The Main Program Starts Here ######

# This portion pulls the data from the webform and places it
# into a string in this format: parm1=value1;parm2=value2;...\0

string = ''                                     # stores commands when not changing observatin
settings
stringTele = ''                                         # stores string of commands that are passed
to telescope camera if setting up observation
stringWide = ''                                         # stores string of commands that are passed
to wide-angle camera if setting up observation
xmlString = ''                                  # stores parameters that get written as XML fields

equipment = ''                                  # stores the piece of equipment that user is
controlling (wide, tele, mount, or observation)
error = '0'                                     # stores an error if there is a problem


# Start printing output to webpage here
print 'Content-type: text/html\n\n'
print '<HTML>\n'

logging.basicConfig( level = logging.DEBUG )
log = logging.getLogger( "control" )
#log.debug( "start" )

form = cgi.FieldStorage()                       # get parameters from webform

#for field in form.keys() :
#       print '%s=%s<br>' % (field,form.getvalue(field))

## Check to make sure that form specified which equipment settings pertain to: ##
try:
        equipment = form.getvalue('equipment')        # if the form field tells which equipment
the instructins are for, grab that information
except:
        error = 'Equipment not specified in web form!'
        errorPrint(error)

# process fields for each form depending on which equipment specified:
        # get fields from webform for each paramter
        # concatenate fields into a string
        # send string to back-end driver

if (equipment == 'mount'):    # if the field is a position setting, these must get stored for now
        operation = form.getvalue('mntOp')    # mntOp tells it which operation it is - it must be
in all mount forms
```

```
	if (operation == 'connect'):                # After Deg, Min, and Sec are acquired for
Azimuth and Altitude,
		string = 'mntConnect=TRUE\r\n\xFF'    # they will be sent to back-end driver in
this format: mntAzi=Deg:Min:Sec\r\n mntAlt=Deg:Min:Sec\r\n
	elif (operation == 'park'):
		string = 'mntConnect=FALSE\r\n\xFF'
	elif (operation == 'abort'):
		string = 'mntAbort=TRUE\r\n\xFF'
	elif (operation == 'home'):
		string = 'mntHome=TRUE\r\n\xFF'
	elif (operation == 'slew'):

		# get user's requested position values to put into string and later write to
mountPrefs.xml
		mntAziDeg = '%s' % (form.getvalue('mntAziDeg'))
		mntAziMin = '%s' % (form.getvalue('mntAziMin'))
		mntAziSec = '%s' % (form.getvalue('mntAziSec'))
		mntAltDeg = '%s' % (form.getvalue('mntAltDeg'))
		mntAltMin = '%s' % (form.getvalue('mntAltMin'))
		mntAltSec = '%s' % (form.getvalue('mntAltSec'))

		string = 'mntAzi=%s:' % (mntAziDeg)          # Put Azimuth in deg:mm:ss
		string = '%s%s:' % (string,mntAziMin)
		string = '%s%s\r\n' % (string,mntAziSec)
		string = '%smntAlt=%s:' % (string,mntAltDeg) # Put Altitude in deg:mm:ss
		string = '%s%s:' % (string,mntAltMin)
		string = '%s%s\r\n' % (string,mntAltSec)
		string = string + 'mntSlewTo=TRUE\r\n\xFF'


elif (equipment == 'teleCam') :
	string = 'camConnect=%s\r\n\xFF' % (form.getvalue('camConnect'))
elif (equipment == 'teleOpt') :
	string = 'optConnect=%s\r\n\xFF' % (form.getvalue('optConnect'))
elif (equipment == 'wideCam') :
	string = 'camConnect=%s\r\n\xFF' % (form.getvalue('camConnect'))
elif (equipment == 'wideOpt') :
	string = 'optConnect=%s\r\n\xFF' % (form.getvalue('optConnect'))

elif (equipment == 'observation') :
	teleSendTo = form.getvalue('teleSendTo')
	wideSendTo = form.getvalue('waSendTo')

	# get the observation parameters; put time into proper format: hh:mm:ss
	obsStartVal = '%s' % form.getvalue('obsStart')

	# if starting an observation, we should pass start and stop times
	if (obsStartVal == 'TRUE') :
		obsSet = ''

		# if valid fields were not entered then send None:None:None for start or stop
times
		# these values originate from the browser if nothing was entered in the field
		# (back-end driver starts immediately or waits until stop is pushed)
		try:
			obsStartTimeHH = '%s' % (form.getvalue('obsStartTimeHH'))
			obsStartTimeMM = '%s' % (form.getvalue('obsStartTimeMM'))
			obsStartTimeSS = '%s' % (form.getvalue('obsStartTimeSS'))

			obsStartTime = '%s:' % (obsStartTimeHH)
			obsStartTime = '%s%s:' % (obsStartTime,obsStartTimeMM)
			obsStartTime = '%s%s' % (obsStartTime,obsStartTimeSS)

		except:
			obsStartTime = 'none'


		try:
			obsStopTimeHH = '%s' % (form.getvalue('obsStopTimeHH'))
			obsStopTimeMM = '%s' % (form.getvalue('obsStopTimeMM'))
			obsStopTimeSS = '%s' % (form.getvalue('obsStopTimeSS'))
```

```
                        obsStopTime = '%s:' % (obsStopTimeHH)
                        obsStopTime = '%s%s:' % (obsStopTime,obsStopTimeMM)
                        obsStopTime = '%s%s' % (obsStopTime,obsStopTimeSS)

                except:
                        obsStopTime = 'none'


                # append the start and stop tiems and obsStart command to the end of the string
                obsSet = 'obsStartTime=%s\r\nobsStopTime=%s\r\nobsStart=TRUE\r\n'%
(obsStartTime,obsStopTime)

        # otherwise just send the stop command
        else :
                obsSet = 'obsStart=FALSE\r\n'


        if (teleSendTo == 'TRUE') :

                # get the telescope specific settings
                teleCamADCrate = '%s' % (form.getvalue('teleCamADCrate'))
                stringTele = 'camADCrate=%s\r\n' % (teleCamADCrate)


                teleCamExpTimeVal = '%s' % (form.getvalue('teleCamExpTime'))              # get
the value and multiply by 1000 to get milliseconds
                try:
                        teleCamExpTime = '%s' % (float(teleCamExpTimeVal) * 1000)
                except:
                        teleCamExpTime = 'None'
                stringTele = '%scamExpTime=%s\r\n' % (stringTele,teleCamExpTime)

                # filter values don't necessearily have to be specified, and if none selected then
field won't show up
                try:
                        teleOptFilter = '%s' % (form.getvalue('teleOptFilter'))
                        teleOptFilter = re.sub(r'[\'\s]','',teleOptFilter)                #
remove tick marks that arise from filter select
                        stringTele = '%soptFilter=%s\r\n' % (stringTele,teleOptFilter)
                except:
                        teleOptFilter = 'none selected'
                        stringTele = stringTele

                teleOptShutter = '%s' % (form.getvalue('teleOptShutter'))
                stringTele = '%soptShutter=%s\r\n' % (stringTele,teleOptShutter)

                # append the observation settings
                stringTele = '%s%s\r\n\xFF' % (stringTele,obsSet)


        if (wideSendTo == 'TRUE') :
                waCamADCrate = '%s' % (form.getvalue('waCamADCrate'))
                stringWide = 'camADCrate=%s\r\n' % (waCamADCrate)

                waCamExpTimeVal = '%s' % (form.getvalue('waCamExpTime'))           # get the
value and multiply by 1000 to get milliseconds
                try:
                        waCamExpTime = '%s' % (float(waCamExpTimeVal) * 1000)
                except:
                        waCamExpTime = 'None'
                stringWide = '%scamExpTime=%s\r\n' % (stringWide,waCamExpTime)

                try:
                        waOptFilter = '%s' % (form.getvalue('waOptFilter'))
                        waOptFilter = re.sub(r'[\'\s]','',waOptFilter)                    #
remove tick marks that arise from filter select
                        stringWide = '%soptFilter=%s\r\n' % (stringWide,waOptFilter)
                except:
                        wideOptFilter = 'none selected'
                        stringWide = stringWide
```

```
                waOptShutter = '%s' % (form.getvalue('waOptShutter'))
                stringWide = '%soptShutter=%s\r\n' % (stringWide,waOptShutter)

                # append the observation settings
                stringWide = '%s%s\r\n\xFF' % (stringWide,obsSet)



### ********************************************************************* ###
# This portion determines the XML file that settings get written to
# It also determines which page to return the user to after processing is done

filename = ''                                   # stores the filename of the User Preferences XML
file to use for the operation
returnPage = ''                                         # stores the page that the user will be re-
directed to after transmission is complete
settings = ''                                   # stores any information that has to get written to
user settings XML file (i.e. it takes whichever string is being used)

if ((equipment == 'teleCam') or (equipment == 'wideCam') or (equipment == 'teleOpt') or
(equipment == 'wideOpt')) :
        filename = obsUserXML
        returnPage = 'observation.html'
elif (equipment == 'observation') :
        filename = obsUserXML
        returnPage = 'observation.html'
elif (equipment == 'mount') :
        filename = mountUserXML
        returnPage = 'mount.html'



### ********************************************************************* ###
#TCP/IP client program sends ASCII string to back-end driver

#determine the IP address(es) to send to
if ((equipment == 'teleCam') or (equipment == 'teleOpt')) :
        sendString(teleCamIP,string)          # send commands to telescope computer
elif ((equipment == 'wideCam') or (equipment == 'wideOpt')) :
        sendString(wideCamIP,string)          # send commands to wide-angle computer
elif (equipment == 'mount') :
        sendString(mountIP,string)            # send commands to mount computer
elif (equipment == 'observation') :
        if (teleSendTo == 'TRUE') :
                sendString(teleCamIP,stringTele)      # send commands to telescope computer
        if (wideSendTo == 'TRUE') :
                sendString(wideCamIP,stringWide)      # send commands to wide-angle computer



### ********************************************************************* ###
#write user preferences to xml so that they can be displayed if an operation is in progress

try:
        #write mount preferences if we are slewing the mount
        if ((equipment == 'mount') and (operation == 'slew')) :
                xmlfile=open(filename,'w')                  #open the XML file for writing
                xmlfile.write('<mountPrefs>\n')
                xmlfile.write('\t<mntAziDeg>' + mntAziDeg + '</mntAziDeg>\n')
                xmlfile.write('\t<mntAziMin>' + mntAziMin + '</mntAziMin>\n')
                xmlfile.write('\t<mntAziSec>' + mntAziSec + '</mntAziSec>\n')
                xmlfile.write('\t<mntAltDeg>' + mntAltDeg + '</mntAltDeg>\n')
                xmlfile.write('\t<mntAltMin>' + mntAltMin + '</mntAltMin>\n')
                xmlfile.write('\t<mntAltSec>' + mntAltSec + '</mntAltSec>\n')
                xmlfile.write('</mountPrefs>')
                xmlfile.close()                                     #close the XML file

        #write observation preferences if we are starting an observation
        elif ((equipment == 'observation') and (obsStartVal == 'TRUE')) :
```

```
                xmlfile=open(filename,'w')                    #open the XML file for writing
                xmlfile.write('<userPrefs>\n')
                xmlfile.write('\t<tele>\n')

                if (teleSendTo == 'TRUE') :
                        xmlfile.write('\t\t<camADCrate>' + teleCamADCrate + '</camADCrate>\n')

                        teleCamExpTime = '%s' % (float(teleCamExpTimeVal))      # write to XML in
seconds (not msec)
                        xmlfile.write('\t\t<camExpTime>' + teleCamExpTime + '</camExpTime>\n')
                        xmlfile.write('\t\t<optFilter>' + teleOptFilter + '</optFilter>\n')
                        xmlfile.write('\t\t<optShutter>' + teleOptShutter + '</optShutter>\n')
                else :
                        xmlfile.write('\t\t<camADCrate>' + 'none' + '</camADCrate>\n')
                        xmlfile.write('\t\t<camExpTime>' + 'none' + '</camExpTime>\n')
                        xmlfile.write('\t\t<optFilter>' + 'none' + '</optFilter>\n')
                        xmlfile.write('\t\t<optShutter>' + 'none' + '</optShutter>\n')

                xmlfile.write('\t</tele>\n')
                xmlfile.write('\t<wide>\n')

                if (wideSendTo == 'TRUE') :
                        xmlfile.write('\t\t<camADCrate>' + waCamADCrate + '</camADCrate>\n')

                        waCamExpTime = '%s' % (float(waCamExpTimeVal))
                        xmlfile.write('\t\t<camExpTime>' + waCamExpTime + '</camExpTime>\n')
                        xmlfile.write('\t\t<optFilter>' + waOptFilter + '</optFilter>\n')
                        xmlfile.write('\t\t<optShutter>' + waOptShutter + '</optShutter>\n')
                else:
                        xmlfile.write('\t\t<camADCrate>' + 'none' + '</camADCrate>\n')
                        xmlfile.write('\t\t<camExpTime>' + 'none' + '</camExpTime>\n')
                        xmlfile.write('\t\t<optFilter>' + 'none' + '</optFilter>\n')
                        xmlfile.write('\t\t<optShutter>' + 'none' + '</optShutter>\n')

                xmlfile.write('\t</wide>\n')
                xmlfile.write('\t<obs>\n')
                xmlfile.write('\t\t<obsStartTime>' + obsStartTime + '</obsStartTime>\n')
                xmlfile.write('\t\t<obsStopTime>' + obsStopTime + '</obsStopTime>\n')
                xmlfile.write('\t</obs>\n')
                xmlfile.write('</userPrefs>\n')
                xmlfile.close()                               #close the XML file


except:
        error = 'Error writing to XML file:' + filename + '\n'
        errorPrint(error)




### ********************************************************************* ###
#This portion tells Apache server to redirect the user back to the referring page

#print 'equipment: ' + equipment + '<br>'
#print 'stringTele: ' + stringTele + '<br>'
#print 'stringWide: ' + stringWide + '<br>'
#print 'string: ' + string

print '<HEAD>\n'
print '<meta http-equiv="REFRESH" content="0;url=' + pageAddress + returnPage + '">\n </HEAD>'
print '<BODY>Processing...</BODY>\n'

print '</HTML>'
#log.debug( "exit" )
sys.exit()
```

Appendix H: Back-end Driver Source Code

BackendDriver.h

```
#ifndef BACKENDDRIVER_
#define BACKENDDRIVER_


/* includes */
#include <iostream>
#include "Observation.h"
#include "TcpIpServer.h"
#include "XmlNode.h"
#include <windows.h>
#include <tchar.h>
#include <strsafe.h>
#include "time.h"

using namespace std;


/* defines */
#define PORT 5000

#define RX_BUF_LEN 1024
#define TX_BUF_LEN 1024
#define TEMP_BUF_LEN 1024

#define DELIMITER   "\n\0"
#define PROMPT "Backend Driver>\0"
#define NEWLINE "\r\n\0"


#define MSG_ACK               "Connected.\r\n\r\n\0"
#define MSG_INVALID_CMD       "Invalid command.\r\n\r\n\0"
#define MSG_CLOSING_CONN      "Closing Connection.\r\n\r\n\0"


#define SLEEP_INTERVAL 5 // seconds

#define IMAGE_BUF_SIZE 512 * 512 * sizeof( uns16 ) * 2 // double size just in case


/**/

/**/
#define WINVIEW_BASE_DIR "winview\0"
#define PNG_BASE_DIR "png\0"


DWORD WINAPI runObservation( LPVOID lpParam );
DWORD WINAPI waitForClientRequest( LPVOID lpParam );
DWORD WINAPI mountOperation( LPVOID lpParam );
int findTimeDifference( ObsTime& a, ObsTime& b );
int checkObservationThreadStatus( Observation* obs, HANDLE* hObservation );
int checkMountOpThreadStatus( Observation* obs, HANDLE *hMountOperation );


#endif
```

BackendDriver.cpp

```
#include "BackendDriver.h"

using namespace std;


int main( )
{
```

```
/* declarations */
TcpIpServer* tcpIpServ; // TCP/IP server object
Observation* obs;       // Observation control object
HANDLE hAcceptClient    = NULL;
HANDLE hObservation     = NULL;
HANDLE hMountOperation  = NULL;
DWORD  dAcceptClient;

char* rxBuf;            // data receive buffer
char* txBuf;            // data transmit buffer
char* tempBuf;          // temporary buffer
bool  quit;             // quit command flag from user

/* initialization */
tcpIpServ = new TcpIpServer( PORT );          // instantiate objects
obs       = new Observation();                // "
rxBuf     = (char *)malloc( RX_BUF_LEN );     // allocate buffers
txBuf     = (char *)malloc( TX_BUF_LEN );     // "
tempBuf   = (char *)malloc( TEMP_BUF_LEN );   // "
memset( txBuf, '\0', TX_BUF_LEN );            // clear buffers
memset( rxBuf, '\0', RX_BUF_LEN );            // "
memset( tempBuf, '\0', TEMP_BUF_LEN );        // "

obs->initializeObs();

/* attempt to initialize the server socket and begin listening for clients */
if( tcpIpServ->initServerAndListen() )
{
  /* unsuccessful */
  cerr << "Couldn't initialize TCP/IP server." << endl;
  return 1;
}

/* begin listening for a request in background */
hAcceptClient = CreateThread(
  NULL,                 // default security attributes
  0,                    // use default stack size
  waitForClientRequest, // thread function
  (LPVOID)tcpIpServ,    // argument to thread function
  0,                    // use default creation flags
  &dAcceptClient);      // returns the thread identifier

/* main program loop */
while(1)
{

  /* limits polling, this loop should be event-driven instead... */
  Sleep( 50 );

  /* check for client connection request... */
  if( !WaitForSingleObject( hAcceptClient, 0 ) )
  {
    /* request received, acknowledge client */
    strcpy_s( txBuf, TX_BUF_LEN, MSG_ACK );
    tcpIpServ->sendToClient( txBuf, (int)strlen(txBuf) );
    /* newline */
    strcpy_s( txBuf, TX_BUF_LEN, NEWLINE );
    tcpIpServ->sendToClient( txBuf, (int)strlen(txBuf) );

    /* while the user has not requested to quit... */
    quit = false;
    while( !quit )
    {

      /* send prompt arrow */
      strcpy_s( txBuf, TX_BUF_LEN, PROMPT );
      tcpIpServ->sendToClient( txBuf, (int)strlen(txBuf) );

      /* clear rx buffer */
      memset( rxBuf, '\0', RX_BUF_LEN );
```

```
/* receive data, if received data is not valid... */
if( tcpIpServ->recvFromClient( rxBuf, RX_BUF_LEN, DELIMITER ) < 0 )
{

  /* notify user */
  strcpy_s( txBuf, TX_BUF_LEN, MSG_INVALID_CMD );
  tcpIpServ->sendToClient( txBuf, (int)strlen(txBuf) );
  /* newline */
  strcpy_s( txBuf, TX_BUF_LEN, NEWLINE );
  tcpIpServ->sendToClient( txBuf, (int)strlen(txBuf) );

  /* quit */
  quit = true;

  /* acknowledge quit message and the close connection */
  strcpy_s( txBuf, TX_BUF_LEN, MSG_CLOSING_CONN );
  tcpIpServ->sendToClient( txBuf, (int)strlen(txBuf) );
  /* newline */
  strcpy_s( txBuf, TX_BUF_LEN, NEWLINE );
  tcpIpServ->sendToClient( txBuf, (int)strlen(txBuf) );
  tcpIpServ->closeClientConn();
  CloseHandle( hAcceptClient );

  /* begin listening for a request in background */
  hAcceptClient = CreateThread(
    NULL,                 // default security attributes
    0,                    // use default stack size
    waitForClientRequest, // thread function
    tcpIpServ,            // argument to thread function
    0,                    // use default creation flags
    &dAcceptClient);      // returns the thread identifier

}
/* if it is valid, process the command */
else
{

  /* check for EOF */
  if( *rxBuf == EOF )
  {
    /* EOF received, user has requested to quit */
    quit = true;

    /* acknowledge quit message and the close connection */
    strcpy_s( txBuf, TX_BUF_LEN, MSG_CLOSING_CONN );
    tcpIpServ->sendToClient( txBuf, (int)strlen(txBuf) );
    /* newline */
    strcpy_s( txBuf, TX_BUF_LEN, NEWLINE );
    tcpIpServ->sendToClient( txBuf, (int)strlen(txBuf) );
    tcpIpServ->closeClientConn();
    CloseHandle( hAcceptClient );

    /* begin listening for a request in background */
    hAcceptClient = CreateThread(
      NULL,                 // default security attributes
      0,                    // use default stack size
      waitForClientRequest, // thread function
      tcpIpServ,            // argument to thread function
      0,                    // use default creation flags
      &dAcceptClient);      // returns the thread identifier
  }

  /* command was not EOF, process it normally */
  else
  {
    /* process the command */
    tempBuf = obs->processCommand( rxBuf );

    /* check for quit command */
    if ( tempBuf == 0 )
```

```
            {
              /* quit */
              quit = true;

              /* acknowledge quit message and the close connection */
              strcpy_s( txBuf, TX_BUF_LEN, MSG_CLOSING_CONN );
              tcpIpServ->sendToClient( txBuf, (int)strlen(txBuf) );
              /* newline */
              strcpy_s( txBuf, TX_BUF_LEN, NEWLINE );
              tcpIpServ->sendToClient( txBuf, (int)strlen(txBuf) );
              tcpIpServ->closeClientConn();
              CloseHandle( hAcceptClient );

              /* begin listening for a request in background */
              hAcceptClient = CreateThread(
                NULL,                 // default security attributes
                0,                    // use default stack size
                waitForClientRequest, // thread function
                tcpIpServ,            // argument to thread function
                0,                    // use default creation flags
                &dAcceptClient);      // returns the thread identifier
            }
            /* non quit command */
            else
            {

              /* output return status to the client */
              memset( txBuf, '\0', TX_BUF_LEN );
              strcpy_s( txBuf, TX_BUF_LEN, tempBuf );
              tcpIpServ->sendToClient( txBuf, (int)strlen(txBuf) );
              /* newline */
              strcpy_s( txBuf, TX_BUF_LEN, NEWLINE );
              tcpIpServ->sendToClient( txBuf, (int)strlen(txBuf) );

              /* cleanup */
              free( tempBuf );

            } // non quit command

          } // command not eof ELSE

        } // command valid ELSE

        checkObservationThreadStatus( obs, &hObservation );
#ifdef _WACAM
        checkMountOpThreadStatus( obs, &hMountOperation );
#endif

      } // while (!quit)

    } // if( !WaitForSingleObject( hAcceptClient, 0 ) )

    checkObservationThreadStatus( obs, &hObservation );
#ifdef _WACAM
    checkMountOpThreadStatus( obs, &hMountOperation );
#endif

  } // while( 1 )

  return 0;

}



int findTimeDifference( ObsTime& a, ObsTime& b )
{
  int total;
  int asec, bsec;
  total = 0;
```

```
  asec = 3600 * a.hours + 60 * a.minutes + a.seconds;
  bsec = 3600 * b.hours + 60 * b.minutes + b.seconds;

  if( bsec > asec )
    return bsec - asec;
  else if ( bsec == asec )
    return 3600 * 24;  // number of seconds in a day
  else
    return asec - bsec;

}

DWORD WINAPI waitForClientRequest( LPVOID lpParam )
{
  TcpIpServer* tcpIpServ;
  tcpIpServ = (TcpIpServer*) lpParam;

  /* waits for a client request */
  if( tcpIpServ->acceptClient() )
    return 1;
  else
    return 0;
}

DWORD WINAPI runObservation( LPVOID lpParam )
{

  /* cast the argument to Observation type */
  Observation* obs;
  obs = (Observation*) lpParam;
  bool doneFlag = false;

  int cameraTemp;
  int currentFilter;
  uns16* image;
  int imageByteSize;
  time_t tempTime;
  time_t timera, timerb, duration;

  /* file path/names */
  char* pngFilepath;
  char* pngFilename;
  char* winviewFilepath;
  char* winviewFilename;
  char* dateAndTime;
  char* command;

  /* allocate memory */
  pngFilepath     = (char *)malloc( FILENAME_LEN );
  pngFilename     = (char *)malloc( FILENAME_LEN );
  winviewFilepath = (char *)malloc( FILENAME_LEN );
  winviewFilename = (char *)malloc( FILENAME_LEN );
  dateAndTime     = (char *)malloc( FILENAME_LEN );
  command         = (char *)malloc( FILENAME_LEN );
  image           = (uns16 *)malloc( IMAGE_BUF_SIZE );

  /* reset the filter sequence */
  obs->Observation_resetFilterSequence();

  /* get the current time */
  time( &tempTime );

  /* use that time to create the directories */
  obs->Observation_generateFilepath( &tempTime, pngFilepath, IMAGE_PNG_BASE_PATH );
  obs->Observation_generateFilepath( &tempTime, winviewFilepath, IMAGE_WINVIEW_BASE_PATH );

  /* create the directories */
  sprintf_s( command, FILENAME_LEN, "mkdir \"%s\"\0", pngFilepath );
  system( command );
  sprintf_s( command, FILENAME_LEN, "mkdir \"%s\"\0", winviewFilepath );
  system( command );
```

```
  /* get the curent time */
  time(&tempTime);
  ObsTime currentTime;
  currentTime.seconds = (int)tempTime % 60;
  currentTime.minutes = (int)tempTime % (60 * 60) / 60;
  currentTime.hours   = (int)tempTime % (60 * 60 * 24) / 3600;
  int timeDifSleep;
  unsigned int currentTimeDifference = 0;
  unsigned int lastTimeDifference = INT_MAX;

  /* should we not start right away? */
  if( obs->getStartTime().hours != -1 )
  {
    /* if not, sleep until we do start */
    timeDifSleep = findTimeDifference( currentTime, obs->getStartTime() );
    cout << "Sleeping for " << timeDifSleep << " seconds." << endl;
    timeDifSleep /= SLEEP_INTERVAL;
    while( timeDifSleep > 0 && !doneFlag )
    {
      Sleep( SLEEP_INTERVAL * 1000 );
      timeDifSleep--;
      if( obs->isObsStopRequested() )
        doneFlag = true;
    }
    cout << "Done sleeping." << endl;
  }
  /* if we were stopped */
  if( doneFlag )
    return 0;


  /* observe until time is up or stop requested */
  doneFlag = false;
  while( !doneFlag )
  {
    time ( &timera );
    /* set the next filter */
    currentFilter = obs->Observation_setNextFilter();

    /* update the camera temperature */
#ifdef _NOHARDWARE
    cameraTemp = -40;
#else
    cameraTemp = obs->Observation_getCameraTemperature();
#endif

    /* perdiodic XML update */
    obs->Observation_periodicXmlUpdate( currentFilter, cameraTemp );
    time ( &timerb );

    duration = timerb - timera;
    cout << "hardware + xml update duration: " << duration << endl;


    /* get the time right before the exposure */
    time( &tempTime );

    /* data acquisition */
#ifdef _NOHARDWARE
    /* simulate picture... */
    Sleep( 5000 );
#else

    time( &timera );
    /* take the picture */
    imageByteSize = obs->Observation_acquireImage( &image );
    time( &timerb );
    duration = timerb - timera;

    cout << "take image time: " << duration << endl;
```

```
    time( &timera );
    /* generate the filenames using the time of the exposure */
    obs->Observation_generateFilename( &tempTime, winviewFilename, currentFilter,
IMAGE_WINVIEW_EXTENSION );
    obs->Observation_generateFilename( &tempTime, pngFilename, currentFilter,
IMAGE_PNG_EXTENSION );

    /* prepare the winview file for writing */
    obs->Observation_openWinviewFile( winviewFilepath, winviewFilename );
    /* save the image to a WinView-openable format */
    obs->Observation_appendImageToWinviewFile( image, imageByteSize );
    /* close the file */
    obs->Observation_closeWinviewFile();

    /* write the PNG */
    obs->Observation_writeImageToPng( image, imageByteSize, pngFilepath, pngFilename );

    obs->Observation_xmlUpdateFilePath( pngFilepath, pngFilename );

    /* delete the image */
    if( image )
      free( image );

    time( &timerb );

    duration = timerb - timera;

    cout << "write file time: " << duration << endl;

#endif


    /* do we have a stop time? */
    if( obs->getStopTime().hours != -1 )
    {
      /* if so, have we reached it? */
      time(&tempTime);
      currentTime.seconds = (int)tempTime % 60;
      currentTime.minutes = (int)tempTime % (60 * 60) / 60;
      currentTime.hours   = (int)tempTime % (60 * 60 * 24) / 3600;
      currentTimeDifference = findTimeDifference( currentTime, obs->getStopTime() );
      if( currentTimeDifference > lastTimeDifference )
        doneFlag = true;
      else
        lastTimeDifference = currentTimeDifference;
    }

    /* has a stop been requested by the user? */
    if( obs->isObsStopRequested() )
      doneFlag = true;
  }

  /* cleanup */
  free( pngFilepath );
  free( pngFilename );
  free( winviewFilepath );
  free( winviewFilename );
  free( dateAndTime );
  free( command );

  return 0;
}


int checkObservationThreadStatus( Observation* obs, HANDLE *hObservation )
{
  DWORD dTest;

  /* if an observation thread is running... */
  if( obs->isObsInProgress() )
```

```
  {
    /* check if done */
    if ( !WaitForSingleObject( *hObservation, 0 ) )
    {
      cerr << "Observation closed." << endl;
      /* close its handle */
      CloseHandle( *hObservation );
      *hObservation = NULL;
      /* clear flags */
      obs->clearObsStopRequest();
      obs->clearObsInProgress();
    }
  }
  else
  {
    if( obs->positionTimer() )
      obs->updateXmlTemperature();
  }

  /* check for observation start request */
  if ( obs->isObsStartRequested() )
  {
    /* observation requested, launch observation thread */
    *hObservation = CreateThread(
      NULL,           // default security attributes
      0,              // use default stack size
      runObservation, // thread function
      obs,            // argument to thread function
      0,              // use default creation flags
      &dTest);        // returns the thread identifier
  }

  return 0;
}


DWORD WINAPI mountOperation( LPVOID lpParam )
{
  /* cast the argument to Observation type */
  Observation* obs;
  obs = (Observation*) lpParam;
  MntPosition alt;
  MntPosition azi;

  switch( obs->getMountOpType() )
  {
  case MOUNT_CONNECT_TYPE:
    if( obs->getMount()->connectMount() )
      obs->getMount()->setFailureStatus();
    else
      obs->updateXmlPosition( obs->getMount()->getPosition() );
    break;

  case MOUNT_HOME_TYPE:
    if( obs->getMount()->homeMount() )
      obs->getMount()->setFailureStatus();
    else
      obs->updateXmlPosition( obs->getMount()->getPosition() );
    break;

  case MOUNT_PARK_TYPE:
    if( obs->getMount()->parkAndDisconnectMount() )
      obs->getMount()->setFailureStatus();
    break;

  case MOUNT_SLEW_TYPE:
    azi = obs->getAzimuth();
    alt = obs->getAltitude();
    if( obs->getMount()->slewTo(  azi.degrees, azi.minutes, (float)azi.minutes,
      alt.degrees, alt.minutes, (float)alt.seconds ) )
      obs->getMount()->setFailureStatus();
```

```
    else
      obs->updateXmlPosition( obs->getMount()->getPosition() );
    break;

  default:
    break;
  }

  return 0;
}


int checkMountOpThreadStatus( Observation* obs, HANDLE *hMountOperation )
{
  DWORD dTest;

  /* if an operations thread is running... */
  if( obs->isMountOpInProgress() )
  {
    /* check if done */
    if ( !WaitForSingleObject( *hMountOperation, 0 ) )
    {
      /* close its handle */
      CloseHandle( *hMountOperation );
      *hMountOperation = NULL;
      /* clear flags */
      obs->clearMountOpInProgress();
    }
  }

  /* check for observation start request */
  if ( obs->isMountOpStartRequested() )
  {
    /* observation requested, launch observation thread */
    *hMountOperation = CreateThread(
      NULL,           // default security attributes
      0,              // use default stack size
      mountOperation, // thread function
      obs,            // argument to thread function
      0,              // use default creation flags
      &dTest);        // returns the thread identifier
  }

  return 0;
}
```

CamCtrl.h

```
#ifndef CAMCTRL_H
#define CAMCTRL_H

#include <windows.h>
#include <iostream>
#include "master.h"
#include "pvcam.h"

using namespace std;

#define IMAGE_FRAME_SIZE 512 * 512 * sizeof(uns16)
#define ADC_FAST  1
#define ADC_SLOW  0
#define DEFAULT_GAIN 2
#define CAMERA_TEMPERATURE -40
#define CAM_INVALID_TEMP -1000
#define MAX_IMAGE_SIZE 512 * 512 * sizeof( int16 )

class CamCtrl {
public:
```

```
  CamCtrl();

  bool connectCam();
  /**
  *   Initializes the PVCAM library and then connects to the camera. Once
  * successfully connected, the temperature of the controller is set to
  * -40 degrees (F) and the shutter is set to open pre-sequence.
  *
  * INPUT:
  *   none
  * OUTPUT:
  *   true on success, false on failure
  */

  bool disconnectCam();

  bool takeImage();
  uns16* getLastImage();
  unsigned int getLastImageByteSize();
  bool setAdcRate(int16 rate);
  bool setFullRegion();
  bool setExposureTime(int16 newExposureTime);

  /* testing */
  bool initImageSequence();
  /**
  *   Performs the necessary operations required to initialize an
  * image sequence.
  *
  * INPUT:
  *   none
  * OUTPUT:
  *   none
  */

  bool uninitImageSequence();
  /**
  *   Performs the necessary operations required to uninitialize an
  * image sequence.
  *
  * INPUT:
  *   none
  * OUTPUT:
  *   none
  */

  bool takeNextImage( uns16* imagePtr );

  bool acquireImage( uns16* imageBuffer );
  /**
  *   Acquires a single image and stores it in the block of memory
  * supplied by the caller.
  *
  * INPUT:
  *   imageBuffer - pointer to block of memory where image will be stored
  *                 must be at least 512 * 512 * 2 bytes large
  * OUTPUT:
  *   true on success, false on failure
  */

  /* accessors */
  char* getCamName();
  int16 readTemp();

private:
  int16     hCam;
  bool      camReady;
  char*     camName;
  uns16*    lastImage;
  uns32     lastImageSize;
  rgn_type* region;
```

```
  int16     exposureTime;
};

#endif
```

CamCtrl.cpp

```
#include "CamCtrl.h"

CamCtrl::CamCtrl()
{
  hCam = NULL;
  camReady = false;
  exposureTime = 0;

  /* allocate memory */
  region = (rgn_type*)malloc( sizeof( rgn_type ) );
  camName = (char *)malloc( CAM_NAME_LEN + 1 );
  memset( camName, 0, CAM_NAME_LEN + 1 );
  lastImage = (uns16 *)malloc( sizeof( 512 * 512 * 2 * 2 ) );
  lastImageSize = 512 * 512 * 2;
}

bool CamCtrl::connectCam()
{
  /* if already connected... */
  if( camReady )
    /* don't bother trying to connect */
    return true;

  if ( !pl_pvcam_init() )
  {
    cerr << "Unable to initialize PVCAM Library." << endl;
    return false;
  }

  if( !pl_cam_get_name( 0, camName ) )
  {
    cerr << "Unable to get camera name." << endl;
    return false;
  }

  if( !pl_cam_open( camName, &hCam, OPEN_EXCLUSIVE ) )
  {
    cerr << "Unable to open camera connection." << endl;
    return false;
  }

  /* set the temperature */
  int16 newTemperature;
  newTemperature = CAMERA_TEMPERATURE * 100;
  if ( !pl_set_param(hCam, PARAM_TEMP_SETPOINT, (void *) &newTemperature) )
  {
    cerr << "Unable to set CCD temperature." << endl;
    return false;
  }

    /* set the shutter state: DISABLED_OPEN */
  uns32 disabledOpen;
  disabledOpen = OPEN_NO_CHANGE;
  if ( !pl_set_param(hCam, PARAM_SHTR_OPEN_MODE, &disabledOpen) )
  {
    cerr << "Unable to set shutter to DISABLED_OPEN." << endl;
    return false;
  }

  /* disable external shutter */
  uns32 notScan;
  notScan = OUTPUT_NOT_SCAN;
  if ( !pl_set_param(hCam, PARAM_LOGIC_OUTPUT, &notScan) )
  {
```

```
    cerr << "Unable to disable external shutter." << endl;
    return false;
  }

camReady = true;

  return true;
}


char* CamCtrl::getCamName()
{
  return camName;
}


bool CamCtrl::disconnectCam()
{
  /* if not connected... */
  if( !camReady )
  {
    cerr << "Camera not connected." << endl;
    /* don't attempt to disconnect */
    return false;
  }

  ///* set the shutter state: NORMAL */
  //uns32 normal;
  //normal = OPEN_PRE_EXPOSURE;
  //if ( !pl_set_param(hCam, PARAM_SHTR_OPEN_MODE, &normal) )
  //{
  //  cerr << "Unable to set shutter to NORMAL." << endl;
  //  return false;
  //}

  if( !pl_cam_close( hCam ) )
  {
    cerr << "Unable to close connection with camera." << endl;
    return false;
  }

  /* no longer connected */
  camReady = false;

  if( !pl_pvcam_uninit() )
  {
    cerr << "Unable to unintialize PVCAM Library." << endl;
    return false;
  }

  return true;
}


bool CamCtrl::setAdcRate(int16 adcRate)
{
  if( !camReady )
  {
    cerr << "Camera not connected." << endl;
    /* camera not connected */
    return false;
  }

  /* set the speed */
  if( !pl_set_param( hCam, PARAM_SPDTAB_INDEX, &adcRate ) )
  {
    cout << "Unable to set ADC speed index." << endl;
    return false;
  }

  return true;
```

```
}

bool CamCtrl::setFullRegion()
{
  if( !camReady )
  {
    cerr << "Camera not connected." << endl;
    /* not connected */
    return false;
  }

  uns16 param;  // temporary parameter

  /* start at first pixel */
  region->s1 = 0;
  region->p1 = 0;

  /* end at last pixel */
  pl_get_param( hCam, PARAM_SER_SIZE, ATTR_DEFAULT, (void *)&param );
  region->s2 = param - 1;
  pl_get_param( hCam, PARAM_PAR_SIZE, ATTR_DEFAULT, (void *)&param );
  region->p2 = param - 1;

  /* binning factors */
  region->sbin = 1;
  region->pbin = 1;

  return true;
}

bool CamCtrl::setExposureTime( int16 newExposureTime )
{
  exposureTime = newExposureTime;
  return true;
}

bool CamCtrl::takeImage()
{

  if( !camReady )
  {
    cerr << "Camera not connected." << endl;
    /* not connected */
    return false;
  }


  /* setup the region */
  setFullRegion();

  /* initialzie the sequence */
  if( !pl_exp_init_seq() )
  {
    cout << "Unable to initialize experiment sequence." << endl;
  }

  /* set up the sequence */
  if( !pl_exp_setup_seq( hCam, 1, 1, region, TIMED_MODE, exposureTime, &lastImageSize ) )
  {
    cout << "Unable to setup sequence." << endl;
  }

  /* allocate space for the image */
  lastImage = (uns16 *)malloc( lastImageSize );

  /* start the sequence */
  pl_exp_start_seq( hCam, lastImage );

  Sleep( exposureTime );

  /* wait for sequence to finish */
```

```
  uns32 not_needed;
  int16 status;
  while( pl_exp_check_status( hCam, &status, &not_needed )
    && (status != READOUT_COMPLETE && status != READOUT_FAILED ) )
    Sleep(50);

  if( status == READOUT_FAILED )
    cout << "Readout failed." << endl;

  /* cleanup */
  pl_exp_finish_seq( hCam, lastImage, 0 );
  pl_exp_uninit_seq();

  return true;
}

uns16* CamCtrl::getLastImage()
{
  return lastImage;
}

unsigned int CamCtrl::getLastImageByteSize()
{
  return (unsigned int)lastImageSize;
}


int16 CamCtrl::readTemp()
{
  if( !camReady )
  {
    cerr << "Camera not connected." << endl;
    /* not connected */
    return CAM_INVALID_TEMP;
  }
  int16 temperature;

  if ( !pl_get_param(hCam, PARAM_TEMP, ATTR_CURRENT, (void *) &temperature) )
  {
    cerr << "Unable to read current CCD temperature." << endl;
  }

  return ( temperature / 100 );
}

bool CamCtrl::initImageSequence()
{

  if( !camReady )
  {
    cerr << "Camera not connected." << endl;
    /* not connected */
    return false;
  }

  /* setup the region (full) */
  setFullRegion();

  /* initialzie the sequence */
  if( !pl_exp_init_seq() )
  {
    cout << "Unable to initialize experiment sequence." << endl;
  }

  /* set up the sequence */
  if( !pl_exp_setup_seq( hCam, 1, 1, region, TIMED_MODE, exposureTime, &lastImageSize ) )
  {
    cout << "Unable to setup sequence." << endl;
  }

  return false;
```

```
}

bool CamCtrl::uninitImageSequence()
{
  /* cleanup */
  pl_exp_finish_seq( hCam, lastImage, 0 );
  pl_exp_uninit_seq();

  return false;
}

bool CamCtrl::takeNextImage( uns16* imagePtr )
{
  /* start the sequence */
  pl_exp_start_seq( hCam, imagePtr );

  /* wait for sequence to finish */
  uns32 not_needed;
  int16 status;
  while( pl_exp_check_status( hCam, &status, &not_needed )
    && (status != READOUT_COMPLETE && status != READOUT_FAILED ) );

  if( status == READOUT_FAILED )
    cout << "Readout failed." << endl;

  return true;
}
```

MountCtrl.h

```
#ifndef MOUNTCTRL_H
#define MOUNTCTRL_H

#include <fstream>
#include <iostream>

using namespace std;

#define MAX_COMMAND_LEN             128
#define FILE_INPUT_LEN              128

#define MOUNT_CONNECT_SUCCESS         "0\0"
#define MOUNT_HOME_SUCCESS            "0\0"
#define MOUNT_PARKDISCONNECT_SUCCESS  "0\0"
#define MOUNT_SLEWTO_SUCCESS          "0\0"
#define MOUNT_ABORT_SUCCESS           "0\0"
#define MOUNT_GETPOSITION_FAILURE     "Mount Failed!\0"

#define MOUNT_CONNECT_FILE            "mount/connect.txt\0"
#define MOUNT_HOME_FILE               "mount/home.txt\0"
#define MOUNT_PARKDISCONNECT_FILE     "mount/park.txt\0"
#define MOUNT_SLEWTO_FILE             "mount/slewto.txt\0"
#define MOUNT_ABORT_FILE              "mount/abort.txt\0"
#define MOUNT_GETPOSITION_FILE        "mount/getposition.txt\0"
#define MOUNT_ISCOMPLETE_FILE         "mount/iscomplete.txt\0"
#define MOUNT_ISCONNECTED_FILE        "mount/isconnected.txt\0"

#define MOUNT_CONNECT_SCRIPT          "mount/connect.vbs\0"
#define MOUNT_HOME_SCRIPT             "mount/home.vbs\0"
#define MOUNT_PARKDISCONNECT_SCRIPT   "mount/park.vbs\0"
#define MOUNT_SLEWTO_SCRIPT           "mount/slewto.vbs\0"
#define MOUNT_ABORT_SCRIPT            "mount/abort.vbs\0"
#define MOUNT_GETPOSITION_SCRIPT      "mount/getposition.vbs\0"
#define MOUNT_ISCOMPLETE_SCRIPT       "mount/iscomplete.vbs\0"
#define MOUNT_ISCONNECTED_SCRIPT      "mount/isconnected.vbs\0"

#define GETPOSITION_DELIM ';'

enum{
  MOUNT_CONNECT_TYPE,
  MOUNT_SLEW_TYPE,
```

```
  MOUNT_HOME_TYPE,
  MOUNT_PARK_TYPE,
  MOUNT_INVALID_TYPE };

  struct posUpdate {
    int azid;
    int azim;
    float azis;
    int altd;
    int altm;
    float alts;
  };
  typedef struct posUpdate PosUpdate;

class MountCtrl
{
public:
  MountCtrl();
  int connectMount();
  int homeMount();
  int parkAndDisconnectMount();
  int slewTo(  int azDeg, int azMin, float azSec,
               int altDeg, int altMin, float altSec );
  int abortOperation();
  PosUpdate* getPosition();

  int setFailureStatus();
  bool testAndClearFailedStatus();


private:
  bool mountReady;
  bool operationFailed;
  long lastPositionUpdate;
  PosUpdate currentPosition;
};


#endif
```

MountCtrl.cpp

```
#include "MountCtrl.h"

MountCtrl::MountCtrl()
{
  mountReady = false;
  operationFailed = false;
  currentPosition.altd = -1;
  currentPosition.altm = -1;
  currentPosition.alts = -1;
  currentPosition.azid = -1;
  currentPosition.azim = -1;
  currentPosition.azis = -1;
}

int MountCtrl::connectMount()
{
  /* don't try to connect if already connected */
  if( mountReady )
    return 0;

  /* error status */
  int errStatus;
  errStatus = 1; // 1 = error

  /* allocate memory for our command string */
  char* command;
  command = (char *)malloc( MAX_COMMAND_LEN );

  /* allocate memory for reading the status file */
```

```
  char* fileInput;
  fileInput = (char *)malloc( FILE_INPUT_LEN );
  memset( fileInput, 0, FILE_INPUT_LEN );

  /* generate the system command */
  sprintf_s(  command,
    MAX_COMMAND_LEN,
    "cscript //nologo %s > %s\0",
    MOUNT_CONNECT_SCRIPT,
    MOUNT_CONNECT_FILE );

  /* call the VBscript to connect to the mount */
  /* the script will write the status of the call to a file */
  system( command );

  /* open the file */
  ifstream statusFile( MOUNT_CONNECT_FILE );
  if( statusFile.is_open() )
  {
    /* for this command, we only need to read the first line of the file */
    statusFile.getline( fileInput, FILE_INPUT_LEN );

    /* test for the "success" indicator */
    if( !strcmp( fileInput, MOUNT_CONNECT_SUCCESS ) )
    {
      errStatus = 0; // successful!
      mountReady = 1;
    }

    /* done with file, closeit */
    statusFile.close();
  }
  else
  {
    /* couldn't open the file, assume command was unsuccessful */
    cerr << "Couldn't open mount connection status file." << endl;
  }

  /* cleanup */
  free( command );
  free( fileInput );

  return errStatus;
}


int MountCtrl::homeMount()
{
  /* don't try to home if not connected */
  if( !mountReady )
    return 1;

  /* error status */
  int errStatus;
  errStatus = 1; // 1 = error

  /* allocate memory for our command string */
  char* command;
  command = (char *)malloc( MAX_COMMAND_LEN );

  /* allocate memory for reading the status file */
  char* fileInput;
  fileInput = (char *)malloc( FILE_INPUT_LEN );
  memset( fileInput, 0, FILE_INPUT_LEN );

  /* generate the system command */
  sprintf_s(  command,
    MAX_COMMAND_LEN,
    "cscript //nologo %s > %s\0",
    MOUNT_HOME_SCRIPT,
    MOUNT_HOME_FILE );
```

```
  /* call the VBscript to connect to the mount */
  /* the script will write the status of the call to a file */
  system( command );

  /* open the file */
  ifstream statusFile( MOUNT_HOME_FILE );
  if( statusFile.is_open() )
  {
    /* for this command, we only need to read the first line of the file */
    statusFile.getline( fileInput, FILE_INPUT_LEN );

    /* test for the "success" indicator */
    if( !strcmp( fileInput, MOUNT_HOME_SUCCESS ) )
      errStatus = 0; // successful!

    /* done with file, closeit */
    statusFile.close();
  }
  else
  {
    /* couldn't open the file, assume command was unsuccessful */
    cerr << "Couldn't home mount." << endl;
  }

  /* cleanup */
  free( command );
  free( fileInput );

  return errStatus;
}


int MountCtrl::parkAndDisconnectMount()
{
  /* don't try to park if not connected */
  if( !mountReady )
    return 1;

  /* error status */
  int errStatus;
  errStatus = 1; // 1 = error

  /* allocate memory for our command string */
  char* command;
  command = (char *)malloc( MAX_COMMAND_LEN );

  /* allocate memory for reading the status file */
  char* fileInput;
  fileInput = (char *)malloc( FILE_INPUT_LEN );
  memset( fileInput, 0, FILE_INPUT_LEN );

  /* generate the system command */
  sprintf_s(  command,
    MAX_COMMAND_LEN,
    "cscript //nologo %s > %s\0",
    MOUNT_PARKDISCONNECT_SCRIPT,
    MOUNT_PARKDISCONNECT_FILE );

  /* call the VBscript to connect to the mount */
  /* the script will write the status of the call to a file */
  system( command );

  /* open the file */
  ifstream statusFile( MOUNT_PARKDISCONNECT_FILE );
  if( statusFile.is_open() )
  {
    /* for this command, we only need to read the first line of the file */
    statusFile.getline( fileInput, FILE_INPUT_LEN );

    /* test for the "success" indicator */
```

```
    if( !strcmp( fileInput, MOUNT_PARKDISCONNECT_SUCCESS ) )
    {
      errStatus = 0; // successful!
      mountReady = false;
    }

    /* done with file, closeit */
    statusFile.close();
  }
  else
  {
    /* couldn't open the file, assume command was unsuccessful */
    cerr << "Couldn't park and disconnect mount." << endl;
  }

  /* cleanup */
  free( command );
  free( fileInput );

  return errStatus;
}

int MountCtrl::slewTo(  int azDeg, int azMin, float azSec,
                        int altDeg, int altMin, float altSec )
{
  /* don't try to slew if not connected */
  if( !mountReady )
    return 1;

  /* error status */
  int errStatus;
  errStatus = 1; // 1 = error

  /* allocate memory for our command string */
  char* command;
  command = (char *)malloc( MAX_COMMAND_LEN );

  /* allocate memory for reading the status file */
  char* fileInput;
  fileInput = (char *)malloc( FILE_INPUT_LEN );
  memset( fileInput, 0, FILE_INPUT_LEN );

  /* generate the system command */
  sprintf_s(  command,
    MAX_COMMAND_LEN,
    "cscript //nologo %s %i %i %2.1f %i %i %2.1f > %s\0",
    MOUNT_SLEWTO_SCRIPT,
    azDeg,
    azMin,
    azSec,
    altDeg,
    altMin,
    altSec,
    MOUNT_SLEWTO_FILE );

  /* call the VBscript to connect to the mount */
  /* the script will write the status of the call to a file */
  system( command );

  /* open the file */
  ifstream statusFile( MOUNT_SLEWTO_FILE );
  if( statusFile.is_open() )
  {
    /* for this command, we only need to read the first line of the file */
    statusFile.getline( fileInput, FILE_INPUT_LEN );

    /* test for the "success" indicator */
    if( !strcmp( fileInput, MOUNT_SLEWTO_SUCCESS ) )
      errStatus = 0; // successful!

    /* done with file, closeit */
```

```
    statusFile.close();
  }
  else
  {
    /* couldn't open the file, assume command was unsuccessful */
    cerr << "Couldn't slew mount." << endl;
  }

  /* cleanup */
  free( command );
  free( fileInput );

  return errStatus;
}


int MountCtrl::abortOperation()
{
  /* don't try to park if not connected */
  if( !mountReady )
    return 1;

  /* error status */
  int errStatus;
  errStatus = 1; // 1 = error

  /* allocate memory for our command string */
  char* command;
  command = (char *)malloc( MAX_COMMAND_LEN );

  /* allocate memory for reading the status file */
  char* fileInput;
  fileInput = (char *)malloc( FILE_INPUT_LEN );
  memset( fileInput, 0, FILE_INPUT_LEN );

  /* generate the system command */
  sprintf_s(  command,
    MAX_COMMAND_LEN,
    "cscript //nologo %s > %s\0",
    MOUNT_ABORT_SCRIPT,
    MOUNT_ABORT_FILE );

  /* call the VBscript to connect to the mount */
  /* the script will write the status of the call to a file */
  system( command );

  /* open the file */
  ifstream statusFile( MOUNT_ABORT_FILE );
  if( statusFile.is_open() )
  {
    /* for this command, we only need to read the first line of the file */
    statusFile.getline( fileInput, FILE_INPUT_LEN );

    /* test for the "success" indicator */
    if( !strcmp( fileInput, MOUNT_ABORT_SUCCESS ) )
    {
      errStatus = 0; // successful!
      mountReady = false;
    }

    /* done with file, closeit */
    statusFile.close();
  }
  else
  {
    /* couldn't open the file, assume command was unsuccessful */
    cerr << "Couldn't abort operation." << endl;
  }

  /* cleanup */
  free( command );
```

```
  free( fileInput );

  return errStatus;
}


PosUpdate* MountCtrl::getPosition()
{
  /* don't try to park if not connected */
  if( !mountReady )
    return 0;

  /* error status */
  int errStatus;
  errStatus = 1; // 1 = error

  /* allocate memory for our command string */
  char* command;
  command = (char *)malloc( MAX_COMMAND_LEN );

  /* allocate memory for reading the status file */
  char* fileInput;
  fileInput = (char *)malloc( FILE_INPUT_LEN );
  memset( fileInput, 0, FILE_INPUT_LEN );

  /* generate the system command */
  sprintf_s(  command,
    MAX_COMMAND_LEN,
    "cscript //nologo %s > %s\0",
    MOUNT_GETPOSITION_SCRIPT,
    MOUNT_GETPOSITION_FILE );

  /* call the VBscript to connect to the mount */
  /* the script will write the status of the call to a file */
  system( command );

  /* open the file */
  ifstream statusFile( MOUNT_GETPOSITION_FILE );
  if( statusFile.is_open() )
  {
    /* for this command, we only need to read the first line of the file */
    statusFile.getline( fileInput, FILE_INPUT_LEN );

    /* test for the "success" indicator */
    if( strcmp( fileInput, MOUNT_GETPOSITION_FAILURE ) )
    {
      errStatus = 0;

      char* temp;
      int i;
      temp = fileInput;
      i = 0;
      /* replace the delimeters with NULL for easier extraction of numbers */
      while( *(temp + i) != '\0' )
      {
        if( *(temp + i) == ';' )
          *(temp + i) = '\0';
        i++;
      }

      /* get data */
      temp = fileInput;
      temp = strchr( temp, '=' );
      currentPosition.azid = atoi(temp+1);
      while( *(temp++) != '\0' );
      temp = strchr( temp, '=' );
      currentPosition.azim = atoi(temp+1);
      while( *(temp++) != '\0' );
      temp = strchr( temp, '=' );
      currentPosition.azis = (float)atoi(temp+1);
      while( *(temp++) != '\0' );
```

```
      temp = strchr( temp, '=' );
      currentPosition.altd = atoi(temp+1);
      while( *(temp++) != '\0' );
      temp = strchr( temp, '=' );
      currentPosition.altm = atoi(temp+1);
      while( *(temp++) != '\0' );
      temp = strchr( temp, '=' );
      currentPosition.alts = (float)atoi(temp+1);

    }

    /* done with file, closeit */
    statusFile.close();
  }
  else
  {
    /* couldn't open the file, assume command was unsuccessful */
    cerr << "Couldn't abort operation." << endl;
  }

  /* cleanup */
  free( command );
  free( fileInput );

  if( errStatus )
    return 0;
  else
    return &currentPosition;
}


int MountCtrl::setFailureStatus()
{
  operationFailed = true;
  return 0;
}

bool MountCtrl::testAndClearFailedStatus()
{
  if( operationFailed )
  {
    operationFailed = false;
    return true;
  }

  return false;
}
```

Observation.h

```
#ifndef OBSERVATION_
#define OBSERVATION_


/***************************** INCLUDES ***********************************/
#include "OpticsCtrl.h"
#include "CamCtrl.h"
#include "MountCtrl.h"
#include "XmlNode.h"
#include <fstream>
#include "lodepng.h"
#include <windows.h>

using namespace std;



/***************************** DEFINES ************************************/

/* buffer lengths */
#define PARAM_BUF_LEN   1024
```

```
#define RET_MSG_LEN 256
#define FILTER_SEQ_LEN  256

#define QUIT_MSG        "QUIT\r\n\0"

/* xml configuration */
#define MOUNT_XML_HEAD_TAG  "mountStatus\0"       // headers for XML file
#define CAM_XML_HEAD_TAG    "camStatus\0"         // "
#define MOUNT_XML_FILE_NAME "mountStatus.xml"     // mount xml filename

/* choose the appropriate camera xml filename */
#ifdef _WACAM
#define CAM_XML_FILE_NAME   "wacamStatus.xml"
#else
#define CAM_XML_FILE_NAME   ".\\telecamStatus.xml"
#endif

#define TIME_DELIM ':'            // hh:mm:ss
#define ALT_AZI_DELIM ':'         // d:m:s
#define OPTICS_COM_PORT 1         // optics connected to COM1

#define PNG_W 512                 // PNG image width
#define PNG_H 512                 // PNG image height
#define PNG_WHITE_CUTOFF 0x02     // PNG high-byte cutoff for white

#ifdef _WACAM
#define IMAGE_CAM_BASE_PATH "wide/\0"
#define FILENAME_PREFIX "WIDEFOV\0"
#else
#define IMAGE_CAM_BASE_PATH "tele/\0"
#define FILENAME_PREFIX "TELE\0"
#endif

#define WINVIEW_HEADER_FILENAME "winview_header"
#define WINVIEW_HEADER_SIZE 4100  // number of bytes in winview header file

#define IMAGE_PNG_EXTENSION "png\0"
#define IMAGE_PNG_BASE_PATH "png/\0"

#define IMAGE_WINVIEW_EXTENSION "spe\0"
#define IMAGE_WINVIEW_BASE_PATH "winview/\0"

#define FILENAME_LEN 128

/****************************** TYPEDEF **********************************/
/* observation time struct */
struct obsTime {
  int hours;
  int minutes;
  int seconds;
};
typedef struct obsTime ObsTime;

/* mount alt/azi struct */
struct mntPosition {
  int degrees;
  int minutes;
  int seconds;
};
typedef struct mntPosition MntPosition;


/************************** OBSERVATION CLASS *****************************/
class Observation
{
public:

  Observation();
  /**
  * A constructor.  Initializes class member variables.
  *
```

```
* INPUT
*   none
* RETURNS
*   nothing
*/


int initializeObs();
/**
* Write me.
*/



char* processCommand( char* cmd );
/**
*   Consumes a command and redirects it to the correct function.  If the
* command is valid, a return message is relayed from the called sub-function
* to the caller of this function.  If the command is not valid, an 'invalid
* command' string is returned.  The commands received are not case-sensitive.
*
* INPUT
*   cmd - The command to be processed.  Must be a cstring in the following
*         format:  "param=value\r....\0".  Everything between the first '\r'
*         and the first '\0' is ignored.
* RETURNS
*     Cstring containing information about the status of the processed command,
*   or an invalid command string if unable to process given command.
*/


int connectOpt();
/**
* Not yet.
*/


int connectCam();
/**
* Not yet.
*/


int connectMount();
/**
* Not yet.
*/


char* optSetShutter( char* value );
/**
*   Sets the optics shutter to the given state.  if an invalid state is
* specified, nothing is done and an invalid command string is returned.
*
* INPUT
*   value - string containing the new state of the shutter.  Must be one of
*           the following:
*             "OPEN"  to open the shutter
*             "CLOSE" to close the shutter
*           anything else is invalid.  These are case sensitive.
* RETURNS
*   Cstring describing the current state of the hardware.
*/


char* optSetFilter( char* value );
/**
* Not yet.
*/
```

```
  char* invalidCommand( char* value );
  /**
  * Not yet.
  */

  char* optConnect( char* value );
  /**
  * Not yet.
  */

  char* obsStart( char* value );
  /**
  * Not yet.
  */

  char* obsStartTime( char* value );
  /**
  * Not yet.
  */

  char* obsStopTime( char* value );
  /**
  * Not yet.
  */

  char* getObsInProgressMsg();
  /**
  *
  */

  char* camConnect( char* value );
  char* camAdcRate( char* value );
  char* camExpTime( char* value );

  char* mntConnect( char* value );
  char* mntHome( char* value );
  char* mntAzi( char* value );
  char* mntAlt( char* value );
  char* mntSlew( char* value );
  char* mntAbort( char* value );

  bool isObsStartRequested();
  bool isObsStopRequested();
  bool isObsInProgress();
  ObsTime getStartTime();
  ObsTime getStopTime();
  void clearObsStopRequest();
  void clearObsStartRequest();
  void clearObsInProgress();
  char* getFilterSequence();

  int Observation_setNextFilter();
  int Observation_resetFilterSequence();
  int Observation_getCameraTemperature();
  int Observation_periodicXmlUpdate( int filter, int temp );
  unsigned int Observation_acquireImage( uns16** image );
  int Observation_openWinviewFile( char* filepath, char* filename );
  int Observation_appendImageToWinviewFile( uns16* image, int size );
  int Observation_closeWinviewFile();
  int Observation_writeImageToPng( uns16* image, int size, char* filepath, char* filename );
  int Observation_xmlUpdateFilePath( char* filepath, char* filename );
  int Observation_generateFilename( time_t* currentTime, char* filename, int currentFilter, const
char* extension );
  int Observation_generateFilepath( time_t* currentTime, char* filepath, const char*
filetypeBasePath );


  int getMountOpType();
  bool isMountOpInProgress();
  bool isMountOpStartRequested();
  int clearMountOpInProgress();
```

```
  OpticsCtrl* getOptics();
  CamCtrl*    getCamera();
  MountCtrl*  getMount();

  MntPosition getAltitude();
  MntPosition getAzimuth();

  bool positionTimer();
  int updateXmlPosition( PosUpdate* pos );
  int updateXmlTemperature( );

private:

  /* hardware control objects */
  OpticsCtrl* optics;
  CamCtrl*    cam;
  MountCtrl*  mnt;

  /* hardware connected */
  bool optReady;
  bool camReady;
  bool mountReady;

  /* observation flags */
  bool obsInProgress;
  bool obsStartRequest;
  bool obsStopRequest;

  /* mount operation flags */
  bool  mountOpInProgress;
  bool  startMountOp;
  int   mountOpType;

  /* xml file generation */
  XmlFields   xmlFields;
  XmlNode*    mountStatus;
  XmlNode*    camStatus;

  /* winview file generation */
  fstream winviewFile;

  /* settings for filename */
  char* myAdcRate;
  int myExposureTime;

  /* string containing sequence of filters to use */
  char* filterSequence;
  int   filterSequenceIndex;
  int   currentFilter;

  /* observation start and stop times */
  ObsTime startTime;
  ObsTime stopTime;

  /* azi / alit */
  MntPosition azimuth;
  MntPosition altitude;

  /* position timer */
  long lastPositionUpdate;

};


/****************************** TYPEDEFS ***********************************/
// declared here because the class must be defined first

/* pointer to member function of Observation: fPtr */
typedef char* (Observation::*fPtr)(char*);
```

```
/* lookup struct containing command and function ptr */
struct lookup {
  char* id;
  fPtr  func;
};
typedef struct lookup Lookup;


#endif
```

Observation.cpp

```
#include "Observation.h"

using namespace std;

/* init lookup table */
#define LOOKUPTABLE_LEN 16
Lookup lookupTable[ LOOKUPTABLE_LEN ] =
{
  /* general */
  { "INVALID"           , &Observation::invalidCommand  },//x

  /* optics */
  { "OPTCONNECT\0"      , &Observation::optConnect      },//x
  { "OPTSHUTTER\0"      , &Observation::optSetShutter   },//x
  { "OPTFILTER\0"       , &Observation::optSetFilter    },//x

  /* camera */
  { "CAMCONNECT\0"      , &Observation::camConnect },//x T/F
  { "CAMEXPTIME\0"      , &Observation::camExpTime },//x sec/ms?
  { "CAMADCRATE\0"      , &Observation::camAdcRate },//x FAST/SLOW

  /* observation */
  { "OBSSTARTTIME\0"    , &Observation::obsStartTime },// hh:mm:ss
  { "OBSSTOPTIME\0"     , &Observation::obsStopTime },// hh:mm:ss
  { "OBSSTART\0"        , &Observation::obsStart },// T/F

  /* mount */
  { "MNTCONNECT\0"      , &Observation::mntConnect },// T/F
  { "MNTHOME\0"         , &Observation::mntHome },// T
  { "MNTAZI\0"          , &Observation::mntAzi },// deg:min:sec
  { "MNTALT\0"          , &Observation::mntAlt },// deg:min:sec
  { "MNTSLEWTO\0"       , &Observation::mntSlew },// T
  { "MNTABORT\0"        , &Observation::mntAbort } // T
};


Observation::Observation()
{
  /* hardware control objects */
  cam     = new CamCtrl();
  optics  = new OpticsCtrl();
  mnt     = new MountCtrl();

  /* no observations in progress */
  obsInProgress   = false;
  obsStartRequest = false;
  obsStopRequest  = false;
  startTime.seconds  = -1;
  startTime.minutes  = -1;
  startTime.hours    = -1;
  stopTime.seconds   = -1;
  stopTime.minutes   = -1;
  stopTime.hours     = -1;

  /* no mount operations in progress */
  mountOpInProgress = false;
  startMountOp = false;
  mountOpType = MOUNT_INVALID_TYPE;
  altitude.degrees  = -1;
```

```
  altitude.minutes  = -1;
  altitude.seconds  = -1;
  azimuth.degrees   = -1;
  azimuth.minutes   = -1;
  azimuth.seconds   = -1;

  filterSequenceIndex = 1;

  filterSequence = (char *)malloc( FILTER_SEQ_LEN );
  myAdcRate = (char *)malloc( 8 );

  memset( filterSequence, '\0', FILTER_SEQ_LEN );
  memset( myAdcRate, '\0', 8 );

  strcpy_s( myAdcRate, 8, "NONE\0" );
  myExposureTime = 0;

  lastPositionUpdate = 0;
}


int Observation::initializeObs()
{

#ifdef _WACAM
  /* mount XML file */
  xmlFields.mntStatus = NULL;
  mountStatus = new XmlNode( MOUNT_XML_HEAD_TAG, &(xmlFields.mntStatus));
  mountStatus->initializeMntTree( xmlFields );
  mountStatus->printTree( MOUNT_XML_FILE_NAME );
#endif

  /* camera XML file */
  xmlFields.camStatus = NULL;
  camStatus = new XmlNode( CAM_XML_HEAD_TAG, &(xmlFields.camStatus));
  camStatus->initializeCamTree( xmlFields );
  camStatus->printTree( CAM_XML_FILE_NAME );

  return 0;

}

bool Observation::isObsStartRequested()
{
  if ( obsStartRequest )
  {
    obsStartRequest = false;
    return true;
  }
  else
    return false;
}

bool Observation::isObsInProgress()
{
  return obsInProgress;
}

bool Observation::isObsStopRequested()
{
  return obsStopRequest;
}
ObsTime Observation::getStartTime()
{
  return startTime;
}

ObsTime Observation::getStopTime()
{
  return stopTime;
}
```

```
void Observation::clearObsStopRequest()
{
  obsStopRequest = false;
}

void Observation::clearObsStartRequest()
{
  obsStartRequest = false;
}
void Observation::clearObsInProgress()
{
  obsInProgress = false;
  /* update XML file */
  sprintf_s( xmlFields.camInProgress, XML_VALUE_LEN, "FALSE\0" );
  camStatus->printTree( CAM_XML_FILE_NAME );
}
char* Observation::processCommand(char* cmd)
{
  /* declarations */
  char* equalsLoc;      // location of first '=' in string
  char* crLoc;          // location of first '\r' in string
  char* retMsg;         // return message to caller;
  char* param = (char*)malloc(PARAM_BUF_LEN);  // command parameter
  char* value = (char*)malloc(PARAM_BUF_LEN);  // command value
  int i, id;

  memset(param, '\0', PARAM_BUF_LEN);
  memset(value, '\0', PARAM_BUF_LEN);

  /* convert the entire command to uppercase */
  i = 0;
  while( *(cmd+i) )
  {
    *(cmd + i) = toupper( *(cmd + i) );
    i++;
  }

  /* test for quit first */
  if ( !strcmp( cmd, QUIT_MSG ) )
    return 0;

  /* find the '=' in the string */
  if ( ( equalsLoc = strchr( cmd, '=' ) ) == NULL)
  {
    /* not found */

    /* call subfunction, get invlid command message */
    retMsg = ( *this.*(lookupTable[0].func) )( value );

    /* cleanup */
    free( param );
    free( value );

    /* relay message to caller */
    return retMsg;
  }

  /* find the '\r' in the string */
  if ( ( crLoc = strchr( cmd, '\r' ) ) == NULL)
  {
    /* not found */

    /* call subfunction, get invlid command message */
    retMsg = ( *this.*(lookupTable[0].func) )( value );

    /* cleanup */
    free( param );
    free( value );

    /* relay message to caller */
```

```
    return retMsg;
  }

  /* extract the parameter name */
  strncpy_s( param, PARAM_BUF_LEN, cmd, equalsLoc - cmd );

  /* cmd now points after '=' */
  cmd = equalsLoc + 1;

  /* extract the value */
  strncpy_s( value, PARAM_BUF_LEN, cmd, crLoc - cmd );

  /* find the param id */
  id = 0;
  for( i = 1; i < LOOKUPTABLE_LEN; i++ )
    if( !strcmp( lookupTable[i].id, param ) )
    {
      id = i;
      break;
    }

    /* call subfunction, get return message */
    retMsg = ( *this.*(lookupTable[id].func) )( value );

    /* cleanup */
    free( param );
    free( value );

    /* relay message to caller */
    return retMsg;
}


char* Observation::getFilterSequence()
{
  return filterSequence;
}

char* Observation::invalidCommand( char* value )
{
  char* retMsg;
  retMsg = (char *)malloc( RET_MSG_LEN );

  sprintf_s( retMsg, RET_MSG_LEN, "Invalid command.\r\n\0" );

  return retMsg;
}

char* Observation::getObsInProgressMsg()
{
  char* retMsg;
  retMsg = (char *)malloc( RET_MSG_LEN );

  sprintf_s( retMsg, RET_MSG_LEN, "Observation in progress.\r\n\0" );

  return retMsg;
}

char* Observation::optConnect( char* value )
{
  /* if an observation is in progress... */
  if ( obsInProgress )
    /* ignore the command */
    return getObsInProgressMsg();

  char* retMsg ;        // return message to the caller
  int state;
  enum {
    CONNECTED,          //  - connect optics
    DISCONNECTED,       //  - disconnect optics
    INV                 //  - invalid command
```

```
  };

  retMsg = (char *)malloc( RET_MSG_LEN );
  state = INV;          // assume invalid state

  /* if the request was to connect */
  if( !strcmp( value, "TRUE\0" ) )
  {
    /* now connected */
    state = CONNECTED;

    /* Don't communicate with hardware if debugging */
#ifndef _NOHARDWARE
    /* open connection (COM 1)*/
    optics->open( OPTICS_COM_PORT );
    /* home filter wheel */
    optics->setPosition( 1 );
    /* close shutter */
    optics->setShutter( 0 );

    /* optics are now 'homed' */
#endif

  }
  /* if the request was to disconnect */
  else if( !strcmp( value, "FALSE\0" ) )
  {
    state = DISCONNECTED;

#ifndef _NOHARDWARE
    /* close connection */
    optics->close();
#endif

  }

  /* select the appropriate return message */
  switch ( state )
  {
  case CONNECTED:
    /* Optics connected */
    sprintf_s( retMsg, RET_MSG_LEN, "Optics connected and homed.\r\n\0" );
    /* update XML file */
    sprintf_s( xmlFields.camOptConnectedState, XML_VALUE_LEN, "TRUE\0" );
    sprintf_s( xmlFields.camShutterState, XML_VALUE_LEN, "CLOSED\0" );
    sprintf_s( xmlFields.camFilterState, XML_VALUE_LEN, "1\0" );
    camStatus->printTree( CAM_XML_FILE_NAME );
    break;

  case DISCONNECTED:
    /* optics disconnected */
    sprintf_s( retMsg, RET_MSG_LEN, "Optics disconnected.\r\n\0" );
    /* update XML file */
    sprintf_s( xmlFields.camOptConnectedState, XML_VALUE_LEN, "FALSE\0" );
    camStatus->printTree( CAM_XML_FILE_NAME );
    break;

  default:
    sprintf_s( retMsg, RET_MSG_LEN, "Invalid command.\r\n\0" );
    break;
  }

  cerr << retMsg;
  /* return the message */
  return retMsg;
}


char* Observation::optSetShutter( char* value )
{
```

```
  /* if an observation is in progress... */
  if ( obsInProgress )
    /* ignore the command */
    return getObsInProgressMsg();

  /* declarations */
  char* retMsg;       // return message to caller
  int state;
  enum{
    OPEN,             //  -shutter open state
    CLOSED,           //  -shutter closed state
    INV               //  -invalid shutter state specified
  };

  /* initialization */
  retMsg = (char *)malloc( RET_MSG_LEN ); // allocate space for them message
  state = INV;                            // assume invalid state at first

  /* if we are debugging the program... */
#ifdef _NOHARDWARE
  /* ... don't attempt to communicate with the hardware */

  /* test for "OPEN" argument first */
  if ( !strcmp( value, "OPEN\0" ) )
  {
    /* select the appropriate shutter state message */
    state = OPEN;
  }
  /* then test for CLOSE */
  else if (!strcmp( value, "CLOSE\0"))
  {
    /* select the appropriate shutter state message */
    state = CLOSED;
  }
  /* neither found */
  else
  {
    /* select the appropriate shutter state message */
    state = INV;
  }

  /* we are not debugging... */
#else
  /* ... communicate with hardware */

  /* test for "OPEN" argument first */
  if ( !strcmp( value, "OPEN\0" ) )
  {
    /* if the 'open shutter' returns 'opened'... */
    if( optics->setShutter(1) == 1 )
      /* select the appropriate shutter state message */
      state = OPEN;
    /* the shutter was not opened... */
    else
      /* select the appropriate shutter state message */
      state = CLOSED;
  }
  /* then test for CLOSE */
  else if (!strcmp( value, "CLOSE\0"))
  {
    /* if the 'close shutter' command returns 'closed'... */
    if( optics->setShutter(0) == 0 )
      /* select the appropriate shutter state message */
      state = CLOSED;
    /* the shutter was not closed... */
    else
      /* select the appropriate shutter state message */
      state = OPEN;
  }
  /* neither found */
  else
```

```
  {
    /* select the appropriate shutter state message */
    state = INV;
  }
#endif

  switch( state )
  {
  case OPEN:
    sprintf_s( retMsg, RET_MSG_LEN, "Shutter state: OPEN.\r\n\0" );
    /* update XML file */
    sprintf_s( xmlFields.camShutterState, XML_VALUE_LEN, "OPEN\0" );
    camStatus->printTree( CAM_XML_FILE_NAME );
    break;

  case CLOSED:
    sprintf_s( retMsg, RET_MSG_LEN, "Shutter state: CLOSED.\r\n\0" );
    /* update XML file */
    sprintf_s( xmlFields.camShutterState, XML_VALUE_LEN, "CLOSED\0" );
    camStatus->printTree( CAM_XML_FILE_NAME );
    break;

  default:
    sprintf_s( retMsg, RET_MSG_LEN, "Invalid shutter state.\r\n\0" );
    break;
  }

  cerr << retMsg;
  /* return the message */
  return retMsg;
}


char* Observation::optSetFilter( char* value )
{
  /*  The input can either be a single digit, such as "1" or an
  * array of digits such as "[2,5,1,...n]".  First we'll check for
  * a single digit, and then for the array.
  */

  /* if an observation is in progress... */
  if ( obsInProgress )
    /* ignore the command */
    return getObsInProgressMsg();

  /* declartions */
  int filterPos;        // current filter position
  int maxFilters;       // max number of filters allowed per observation
  char* retMsg;         // return message to caller
  int state;            // state used to select return message
  enum {
    ONE_FILT,           //  -one filter specified state
    FILT_SEQ,           //  -filter sequence state
    INV                 //  -invalid input state
  };

  /* init */
  maxFilters = 6;       // six filters allowed per observation
  filterPos = 0;        // start out with an invalid filter position
  state = INV;          // assume invalid state
  retMsg = (char *)malloc( RET_MSG_LEN ); // allocate space for them message


  /* check for a digit first */
  if ( *value >= '0' && *value <= '9' )
  {
    /* digit found, one filter change is assumed */
    state = ONE_FILT;

    /* read the new filter position from the argument */
    filterPos = *value - 0x30;
```

```
    /* no sequence detected, clear the sequence */
    memset( filterSequence, '\0', FILTER_SEQ_LEN );

    /* copy the single filter into the sequence */
    *filterSequence = *value;

    /* if we are not debugging... */
#ifndef _NOHARDWARE
    /* ...attempt to set the filter wheel position */
    filterPos = optics->setPosition( filterPos );
#endif

  }

  /* no digit found, check for '[' */
  else if( *value == '[' )
  {
    /* possible sequence found, cycle through the array and check each value */
    int i;                  // iteration variable
    char* temp;             // ptr to current position in input array,
    temp = value + 1;       // skip '[' and start at first digit
    for( i = 0; i < maxFilters; i++ )
    {
      /* if we find a digit... */
      if ( *temp >= '0' && *temp <= '9' )
      {
        temp++;
        /* followed by a ',' ... */
        if ( *temp == ',' )
        {
          temp++;
          /* then we have a valid entry, proceed to the next */
        }

        /* or followed by a ']' ... */
        else if( *temp == ']' )
        {
          /* we have a valid sequence of filter numbers */
          state = FILT_SEQ;

          /* null terminate the string */
          *(temp+1) = '\0';

          /* set the sequence */
          strcpy_s( filterSequence, FILTER_SEQ_LEN, value );

          /* extract first filter setting */
          filterPos = *(value + 1) - 0x30;

          /* if we are not debugging... */
#ifndef _NOHARDWARE
          /* ...attempt to set the filter wheel position (first in seq) */
          filterPos = optics->setPosition( filterPos );
#endif
          /* stop searching */
          i = maxFilters + 1;
        }
        /* invalid entry */
        else
          /* stop searching */
          i = maxFilters + 1;
      }
      /* invalid entry */
      else
        /* stop searching */
        i = maxFilters + 1;
    }
  }
```

```
  /* choose the correct return message using the state */
  switch ( state )
  {
  case ONE_FILT:
    /* one filter, no sequence */
    sprintf_s(  retMsg,
      RET_MSG_LEN,
      "Filter set to: %i.\r\nNo filter sequence specified.\r\n\0",
      filterPos
      );
    cerr << retMsg;
    /* update XML file */
    sprintf_s( xmlFields.camFilterState, XML_VALUE_LEN, "%i\0", filterPos );
    camStatus->printTree( CAM_XML_FILE_NAME );
    break;

  case FILT_SEQ:
    /* sequence */
    sprintf_s(  retMsg,
      RET_MSG_LEN,
      "Filter set to: %i.\r\nFilter sequence specified: %s.\r\n\0",
      filterPos,
      filterSequence
      );
    cerr << retMsg;
    /* update XML file */
    sprintf_s( xmlFields.camFilterState, XML_VALUE_LEN, "%i\0", filterPos );
    camStatus->printTree( CAM_XML_FILE_NAME );
    break;

  default:
    /* bad command */
    sprintf_s(  retMsg,
      RET_MSG_LEN,
      "Invalid filter/sequence specified.\r\n\0"
      );
    cerr << retMsg;
  }

  cerr << retMsg;
  /* return the message */
  return retMsg;
}

char* Observation::obsStart( char* value )
{
  char* retMsg ;        // return message to the caller
  int state;
  enum {
    START,              //  - start observation
    STOP,               //  - stop observation
    ALREADY_IN_PROG,    //  - observation already in progress
    NOT_IN_PROG,        //  - observation not in progress
    INV                 //  - invalid command
  };

  retMsg = (char *)malloc( RET_MSG_LEN );
  state = INV;          // assume invalid state

  /* if the request was to start an observation */
  if( !strcmp( value, "TRUE\0" ) )
  {
    /* if an observation is not in progress... */
    if( !obsInProgress )
    {
      /* signal the start of an observation */
      state = START;
      obsStartRequest = true;
      obsInProgress = true;
    }
    /* if one is already in progress, notify the user */
```

```
    else
    {
      state = ALREADY_IN_PROG;
    }
  }
  /* if the request was to stop the observation*/
  else if( !strcmp( value, "FALSE\0" ) )
  {
    /* if an observation is in progress */
    if( obsInProgress )
    {
      /* set the state */
      state = STOP;
      /* request to stop it */
      obsStopRequest = true;
    }
    /* if one is not currently in progress, notify the user */
    else
    {
      state = NOT_IN_PROG;
    }
  }

  /* select the appropriate return message */
  switch ( state )
  {
  case START:
    /* Observation start requested */
    sprintf_s( retMsg, RET_MSG_LEN, "Observation started.\r\n\0" );
    /* update XML file */
    sprintf_s( xmlFields.camInProgress, XML_VALUE_LEN, "TRUE\0" );
    camStatus->printTree( CAM_XML_FILE_NAME );
    break;

  case ALREADY_IN_PROG:
    /* Observation start requested, but one is already in progress */
    sprintf_s( retMsg, RET_MSG_LEN, "Observation already in progress.\r\n\0" );
    break;

  case STOP:
    /* Observation stop requested */
    sprintf_s( retMsg, RET_MSG_LEN, "Halting observation.\r\n\0" );
    break;

  case NOT_IN_PROG:
    /* Observation stop requested, but no observation is running */
    sprintf_s( retMsg, RET_MSG_LEN, "No observation in progress.\r\n\0" );
    break;

  default:
    sprintf_s( retMsg, RET_MSG_LEN, "Invalid command.\r\n\0" );
    break;
  }

  cerr << retMsg;
  /* return the message */
  return retMsg;
}


char* Observation::camAdcRate( char* value )
{
  /* if an observation is in progress... */
  if ( obsInProgress )
    /* ignore the command */
    return getObsInProgressMsg();

  char* retMsg ;        // return message to the caller
  int state;            // i forget
  enum {
    FAST,               //  - fast rate
```

```
    SLOW,               //  - slow rate
    INV                 //  - invalid command
  };

  retMsg = (char *)malloc( RET_MSG_LEN );
  state = INV;          // assume invalid state

  /* fast */
  if( !strcmp( value, "FAST\0" ) )
  {
    /* now connected */
    state = FAST;
    strcpy_s( myAdcRate, 8, value );

#ifndef _NOHARDWARE
    cam->setAdcRate(ADC_FAST);
#endif
  }
  /* slow */
  else if( !strcmp( value, "SLOW\0" ) )
  {
    /* now connected */
    state = SLOW;
    strcpy_s( myAdcRate, 8, value );

#ifndef _NOHARDWARE
    cam->setAdcRate(ADC_SLOW);
#endif
  }

  /* select the appropriate return message */
  switch ( state )
  {
  case FAST:
    /* fast */
    sprintf_s( retMsg, RET_MSG_LEN, "Camera ADC rate set to FAST.\r\n\0" );
    break;

  case SLOW:
    /* slow */
    sprintf_s( retMsg, RET_MSG_LEN, "Camera ADC rate set to SLOW.\r\n\0" );
    break;

  default:
    sprintf_s( retMsg, RET_MSG_LEN, "Invalid command.\r\n\0" );
    break;
  }

  cerr << retMsg;
  /* return the message */
  return retMsg;
}

char* Observation::camConnect( char* value )
{
  /* if an observation is in progress... */
  if ( obsInProgress )
    /* ignore the command */
    return getObsInProgressMsg();

  char* retMsg ;        // return message to the caller
  int state;
  enum {
    CONNECTED,
    DISCONNECTED,
    INV
  };

  retMsg = (char *)malloc( RET_MSG_LEN );
  state = INV;          // assume invalid state
```

```
  /* connect */
  if( !strcmp( value, "TRUE\0" ) )
  {
    /* now connected */
    state = CONNECTED;

#ifndef _NOHARDWARE
    cam->connectCam();
#endif
  }
  /* disconnect */
  else if( !strcmp( value, "FALSE\0" ) )
  {
    /* not connected */
    state = DISCONNECTED;

#ifndef _NOHARDWARE
    cam->disconnectCam();
#endif
  }

  /* select the appropriate return message */
  switch ( state )
  {
  case CONNECTED:
    sprintf_s( retMsg, RET_MSG_LEN, "Camera connected.\r\n\0" );
    /* update XML file */
    sprintf_s( xmlFields.camConnectedState, XML_VALUE_LEN, "TRUE\0" );
    camStatus->printTree( CAM_XML_FILE_NAME );
    break;

  case DISCONNECTED:
    sprintf_s( retMsg, RET_MSG_LEN, "Camera disconnected.\r\n\0" );
    /* update XML file */
    sprintf_s( xmlFields.camConnectedState, XML_VALUE_LEN, "FALSE\0" );
    camStatus->printTree( CAM_XML_FILE_NAME );
    break;

  default:
    sprintf_s( retMsg, RET_MSG_LEN, "Invalid command.\r\n\0" );
    break;
  }

  cerr << retMsg;
  /* return the message */
  return retMsg;
}


char* Observation::camExpTime( char* value )
{
  /* if an observation is in progress... */
  if ( obsInProgress )
    /* ignore the command */
    return getObsInProgressMsg();

  char* retMsg ;        // return message to the caller
  int state;
  int temp;

  enum {
    VALID,              // valid input
    INV                 // invalid
  };

  retMsg = (char *)malloc( RET_MSG_LEN );
  state = INV;          // assume invalid state

  /* convert the number */
  temp = atoi( value );
  /* is it a valid number? */
```

```
  if( temp > 0 )
  {
    state = VALID;
    myExposureTime = temp / 1000; // ms to sec
#ifndef _NOHARDWARE
    cam->setExposureTime( temp );
#endif
  }

  /* select the appropriate return message */
  switch ( state )
  {
  case VALID:
    sprintf_s( retMsg, RET_MSG_LEN, "Exposure time set to: %i.\r\n\0", temp );
    break;

  default:
    sprintf_s( retMsg, RET_MSG_LEN, "Invalid command.\r\n\0" );
    break;
  }

  cerr << retMsg;
  /* return the message */
  return retMsg;
}


char* Observation::mntConnect( char* value )
{
#ifndef _WACAM
  return invalidCommand( value );
#endif

  /* if an observation is in progress... */
  if ( obsInProgress )
    /* ignore the command */
    return getObsInProgressMsg();

  char* retMsg ;        // return message to the caller
  int state;
  enum {
    CONNECTED,
    DISCONNECTED,
    INV
  };

  retMsg = (char *)malloc( RET_MSG_LEN );
  state = INV;          // assume invalid state

  /* c */
  if( !strcmp( value, "TRUE\0" ) )
  {
    /* now connected */
    state = CONNECTED;

#ifndef _NOHARDWARE
    mountOpInProgress = true;
    startMountOp = true;
    mountOpType = MOUNT_CONNECT_TYPE;
#endif
  }
  /* d/c */
  else if( !strcmp( value, "FALSE\0" ) )
  {
    /* not connected */
    state = DISCONNECTED;

#ifndef _NOHARDWARE
    mountOpInProgress = true;
    startMountOp = true;
    mountOpType = MOUNT_PARK_TYPE;
```

```
#endif
  }

  /* select the appropriate return message */
  switch ( state )
  {
  case CONNECTED:
    sprintf_s( retMsg, RET_MSG_LEN, "Mount connected.\r\n\0" );
    /* update XML file */
    sprintf_s( xmlFields.mntSlewingState, XML_VALUE_LEN, "TRUE\0" );
    mountStatus->printTree( MOUNT_XML_FILE_NAME );
    break;

  case DISCONNECTED:
    sprintf_s( retMsg, RET_MSG_LEN, "Parking and disconnecting mount.\r\n\0" );
    /* update XML file */
    sprintf_s( xmlFields.mntSlewingState, XML_VALUE_LEN, "TRUE\0" );
    mountStatus->printTree( MOUNT_XML_FILE_NAME );
    break;

  default:
    sprintf_s( retMsg, RET_MSG_LEN, "Invalid command.\r\n\0" );
    break;
  }

  cerr << retMsg;
  /* return the message */
  return retMsg;
}

char* Observation::obsStartTime( char* value )
{

  /* if an observation is in progress... */
  if ( obsInProgress )
    /* ignore the command */
    return getObsInProgressMsg();

  char* retMsg ;        // return message to the caller
  char* currentNumber;  // string containing the current number
  int state;            // i forget
  enum {
    VALID,              //  - valid time specified
    INV                 //  - invalid command
  };

  retMsg = (char *)malloc( RET_MSG_LEN );
  currentNumber = (char *)malloc( RET_MSG_LEN );
  state = INV;          // assume invalid state

  /* delimeter positions */
  char *delimPos1, *delimPos2;
  /* hours, minutes, seconds */
  int h, m, s;

  /* make sure the string consists of only digits and delimeters */
  char* temp = value;
  bool isValid = true;
  while( *temp != '\0' )
  {
    if( !isdigit(*temp) && ( *temp != TIME_DELIM ) )
      isValid = false;
    temp++;
  }

  if( (delimPos1 = strchr( value, TIME_DELIM ) ) )
    if( (delimPos2 = strchr( delimPos1+1, TIME_DELIM ) ) )
    {
      /* TODO: test for all numeric chars 1:a:1 will result in 1:0:1 */

      /* attempt to extract the hours */
```

```
      memset( currentNumber, 0, RET_MSG_LEN );
      strncpy_s( currentNumber, RET_MSG_LEN, value, (delimPos1 - value) );
      h = atoi( currentNumber );

      /* attempt to extract the minutes */
      memset( currentNumber, 0, RET_MSG_LEN );
      strncpy_s( currentNumber, RET_MSG_LEN, delimPos1 + 1, (delimPos2 - delimPos1) );
      m = atoi( currentNumber );

      /* attempt to extract the seconds */
      memset( currentNumber, 0, RET_MSG_LEN );
      strncpy_s( currentNumber, RET_MSG_LEN, delimPos2 + 1, RET_MSG_LEN );
      s = atoi( currentNumber );

      /* validate values */
      if( h >= 0 && h <= 23 && isValid)
        if( m >= 0 && m <= 59 )
          if( s >= 0 && s <= 59 )
            state = VALID;

      if( state != VALID )
      {
        h = -1; m = -1; s = -1;
      }

    }

    /* select the appropriate return message */
    switch ( state )
    {
    case VALID:
      /* fast */
      sprintf_s( retMsg, RET_MSG_LEN, "Observation start time set to %i:%i:%i.\r\n\0", h, m, s );
      startTime.hours = h; startTime.minutes = m; startTime.seconds = s;
      break;

    default:
      sprintf_s( retMsg, RET_MSG_LEN, "Invalid command.\r\n\0" );
      break;
    }

    cerr << retMsg;
    /* return the message */
    return retMsg;
}


char* Observation::obsStopTime( char* value )
{

  /* if an observation is in progress... */
  if ( obsInProgress )
    /* ignore the command */
    return getObsInProgressMsg();

  char* retMsg ;        // return message to the caller
  char* currentNumber;  // string containing the current number
  int state;            // i forget
  enum {
    VALID,              //  - valid time specified
    INV                 //  - invalid command
  };

  retMsg = (char *)malloc( RET_MSG_LEN );
  currentNumber = (char *)malloc( RET_MSG_LEN );
  state = INV;          // assume invalid state

  /* delimeter positions */
  char *delimPos1, *delimPos2;
  /* hours, minutes, seconds */
  int h, m, s;
```

```
  /* make sure the string consists of only digits and delimeters */
  char* temp = value;
  bool isValid = true;
  while( *temp != '\0' )
  {
    if( !isdigit(*temp) && ( *temp != TIME_DELIM ) )
      isValid = false;
    temp++;
  }

  if( (delimPos1 = strchr( value, TIME_DELIM ) ) )
    if( (delimPos2 = strchr( delimPos1+1, TIME_DELIM ) ) )
    {

      /* attempt to extract the hours */
      memset( currentNumber, 0, RET_MSG_LEN );
      strncpy_s( currentNumber, RET_MSG_LEN, value, (delimPos1 - value) );
      h = atoi( currentNumber );

      /* attempt to extract the minutes */
      memset( currentNumber, 0, RET_MSG_LEN );
      strncpy_s( currentNumber, RET_MSG_LEN, delimPos1 + 1, (delimPos2 - delimPos1) );
      m = atoi( currentNumber );

      /* attempt to extract the seconds */
      memset( currentNumber, 0, RET_MSG_LEN );
      strncpy_s( currentNumber, RET_MSG_LEN, delimPos2 + 1, RET_MSG_LEN );
      s = atoi( currentNumber );

      /* validate values */
      if( h >= 0 && h <= 23 && isValid)
        if( m >= 0 && m <= 59 )
          if( s >= 0 && s <= 59 )
            state = VALID;

      if( state != VALID )
      {
        h = -1; m = -1; s = -1;
      }

    }

    /* select the appropriate return message */
    switch ( state )
    {
    case VALID:
      /* valid */
      sprintf_s( retMsg, RET_MSG_LEN, "Observation stop time set to %i:%i:%i.\r\n\0", h, m, s );
      stopTime.hours = h; stopTime.minutes = m; stopTime.seconds = s;
      break;

    default:
      sprintf_s( retMsg, RET_MSG_LEN, "Invalid command.\r\n\0" );
      break;
    }

    cerr << retMsg;
    /* return the message */
    return retMsg;
}


OpticsCtrl* Observation::getOptics()
{
  return optics;
}

CamCtrl* Observation::getCamera()
{
  return cam;
```

```
}

int Observation::Observation_setNextFilter()
{
  int currentFilter;

  /* are we using more than one filter? */
  if( !isdigit( *filterSequence ) )
  {
    /* if so, extract the current filter */
    currentFilter = *(filterSequence + filterSequenceIndex) - 0x30;

    /* are there any more left in the sequence? */
    if( *( filterSequence + filterSequenceIndex + 1 ) == ',' )
      /* yes, point to the next one */
      filterSequenceIndex += 2;
    else
      /* no, start over */
      filterSequenceIndex = 1;
  }
  else
    /* just one filter is being used */
    currentFilter = *filterSequence - 0x30;

#ifndef _NOHARDWARE
  /* now change the filter */
  currentFilter = optics->setPosition( currentFilter );
#endif

  return currentFilter;
}
int Observation::Observation_resetFilterSequence()
{
  filterSequenceIndex = 1;
  return 0;
}

int Observation::Observation_periodicXmlUpdate( int filter, int temp )
{
  sprintf_s( xmlFields.camFilterState, XML_VALUE_LEN, "%i\0", filter );
  sprintf_s( xmlFields.camCcdTempState, XML_VALUE_LEN, "%i\0", temp );
  camStatus->printTree( CAM_XML_FILE_NAME );

  return 0;
}

int Observation::Observation_xmlUpdateFilePath( char* filepath, char* filename )
{
  sprintf_s( xmlFields.camLastPicName, XML_VALUE_LEN, "%s\0", filename );
  sprintf_s( xmlFields.camLastPicPath, XML_VALUE_LEN, "%s\0", filepath );
  camStatus->printTree( CAM_XML_FILE_NAME );

  return 0;
}


int Observation::Observation_getCameraTemperature()
{
  return (int)cam->readTemp();
}

unsigned int Observation::Observation_acquireImage( uns16** image )
{
  cam->takeImage();
  *image = cam->getLastImage();
  return cam->getLastImageByteSize();
}

int Observation::Observation_openWinviewFile( char* filepath, char* filename )
{
  /* complete filename */
```

```
  char* completePath;
  completePath = (char *)malloc( FILENAME_LEN );

  /* combine the path and name into one */
  strcpy_s( completePath, FILENAME_LEN, filepath );
  strcat_s( completePath, FILENAME_LEN, filename );

  /* open the file in binary mode */
  winviewFile.open( completePath, ios::binary | ios::out );

  /* APPEND THE HEADER */
  FILE* pFile;
  char* header;

  fopen_s( &pFile, WINVIEW_HEADER_FILENAME, "rb" );
  header = (char *)malloc( WINVIEW_HEADER_SIZE );
  fread( header, 1, WINVIEW_HEADER_SIZE, pFile );

  header[42]   = 0;
  header[43]   = 2;
  header[656]  = 0;
  header[657]  = 2;
  header[108]  = 3;
  header[109]  = 0;
  header[1446] = 1;
  header[1447] = 0;
  header[1448] = 0;
  header[1449] = 0;

  winviewFile.write( header, WINVIEW_HEADER_SIZE );

  fclose( pFile );
  free( header );
  free( completePath );
  return 0;
}

int Observation::Observation_appendImageToWinviewFile( uns16* image, int size )
{
  winviewFile.write( (char *) image, size );
  return 0;
}

int Observation::Observation_closeWinviewFile()
{
  winviewFile.close();
  return 0;
}

int Observation::Observation_writeImageToPng( uns16* image, int size, char* filepath, char*
filename )
{
  /* complete filename */
  char* completePath;
  completePath = (char *)malloc( FILENAME_LEN );

  /* combine the path and name into one */
  strcpy_s( completePath, FILENAME_LEN, filepath );
  strcat_s( completePath, FILENAME_LEN, filename );

  int i, j, temp;
  unsigned int* histogram;
  int count;
  unsigned short lowP, highP;

  unsigned char* cPtr;
  unsigned char* cImage;
  cImage = (unsigned char*)malloc( PNG_W * PNG_H );
  histogram = (unsigned int*)malloc( sizeof(int) * 256 );
  memset( histogram, 0, sizeof(int) * 256 );
```

```
LodePNG::Encoder encoder;

encoder.infoRaw.color.colorType = 0;
encoder.infoRaw.color.bitDepth  = 8;
encoder.infoPng.color.colorType = 0;
encoder.infoPng.color.bitDepth  = 8;

encoder.getSettings().zlibsettings.useLZ77 = true;
encoder.getSettings().zlibsettings.windowSize = 32768;
encoder.getSettings().zlibsettings.btype = 0;

cPtr = (unsigned char*)image;
j = 0;
for( i = 0; i < size; i+=2 )
  /* if the high byte of the short is above our white cutoff level... */
  if( cPtr[i+1] > PNG_WHITE_CUTOFF )
    /* set the pixel to white */
    cImage[j++] = 255;
  else
    /* if not just extract the low byte */
    cImage[j++] = cPtr[i];

/* populate the histogram */
for( i = 0; i < PNG_W * PNG_H; i++ )
  histogram[cImage[i]]++;

/* calculate 5%, 95% */
i = 1;
count = 0;
while( count < (PNG_W * PNG_H * 0.05) )
{
  lowP = i;
  count += histogram[i++];
  if( i > 255 )
  {
    lowP = 0;
    break;
  }
}


i = 255;
count = 0;
while( count < (PNG_W * PNG_H * 0.05) )
{
  highP = i;
  count += histogram[i--];
  if( i > 255 )
  {
    highP = 255;
    break;
  }
}

/* apply the contrast change */
for( i = 0; i < PNG_W * PNG_H; i++ )
{
  if( cImage[i] > highP )
    cImage[i] = 255;
  else if( cImage[i] < lowP )
    cImage[i] = 0;
  else
  {
    temp = ( cImage[i] - lowP ) * 255;
    if( highP != lowP ) // don't divide by zero
      temp /= (highP - lowP);
    cImage[i] = (unsigned char)temp;
  }
}

/* ecode and save the data */
```

```
  std::vector<unsigned char> buffer;
  encoder.encode( buffer, (unsigned char*)cImage, PNG_W, PNG_H );
  LodePNG::saveFile( buffer, completePath );

  /* free memory */
  free( completePath );
  free( cImage );
  free( histogram );

  return 0;
}

int Observation::Observation_generateFilename( time_t* currentTime, char* filename, int
currentFilter, const char* extension )
{
  char* dateAndTime;    // string containing the date and time
  dateAndTime = (char *)malloc( 128 );

  /* elements in date/time string */
  char* year;
  char* month;
  char* day_str;
  char* day_num;
  char* hour;
  char* min;
  char* sec;

  /* get the date/time string */
  ctime_s( dateAndTime, 128, currentTime );

  /* the string is in the following format */
  /* DDD MMM DD HH:MM:SS YYYY\n\0 */
  /* Wed Jan 02 02:03:55 1980\n\0 */
  /* the field widths remain constant so we can assume we know where they are */

  /* set the chars after our values to null for easy cstring access */
  *(dateAndTime + 3 ) = 0;
  *(dateAndTime + 7 ) = 0;
  *(dateAndTime + 10) = 0;
  *(dateAndTime + 13) = 0;
  *(dateAndTime + 16) = 0;
  *(dateAndTime + 19) = 0;
  *(dateAndTime + 24) = 0;

  /* copy the values to the sub strings */
  year = dateAndTime + 20;
  month = dateAndTime + 4;
  day_str = dateAndTime + 0;
  day_num = dateAndTime + 8;
  hour = dateAndTime + 11;
  min = dateAndTime + 14;
  sec = dateAndTime + 17;

  /* compile the string */
  /* PREFIX_YYYY-MM-DD_HH_MM_SS_FILTERNUM_ADCRATE_EXPOSURE(sec).EXT\0 */
  sprintf_s(  filename,
    FILENAME_LEN,
    "%s_%s-%s-%s_%s_%s_%s_filt%i_%s_%is.%s\0",
    FILENAME_PREFIX,
    year,
    month,
    day_num,
    hour,
    min,
    sec,
    currentFilter,
    myAdcRate,
    myExposureTime,
    extension );

  /* deallocate memory */
```

```
  if( dateAndTime ) free (dateAndTime);

  return 0;
}



int Observation::Observation_generateFilepath( time_t* currentTime, char* filepath, const char*
filetypeBasePath )
{
  char* dateAndTime;    // string containing the date and time
  dateAndTime = (char *)malloc( 128 );

  /* elements in date/time string */
  char* year;
  char* month;
  char* day_str;
  char* day_num;
  char* hour;
  char* min;
  char* sec;

  /* get the date/time string */
  ctime_s( dateAndTime, 128, currentTime );

  /* the string is in the following format */
  /* DDD MMM DD HH:MM:SS YYYY\n\0 */
  /* Wed Jan 02 02:03:55 1980\n\0 */
  /* the field widths remain constant so we can assume we know where they are */

  /* set the chars after our values to null for easy cstring access */
  *(dateAndTime + 3 ) = 0;
  *(dateAndTime + 7 ) = 0;
  *(dateAndTime + 10) = 0;
  *(dateAndTime + 13) = 0;
  *(dateAndTime + 16) = 0;
  *(dateAndTime + 19) = 0;
  *(dateAndTime + 24) = 0;

  /* copy the values to the sub strings */
  year = dateAndTime + 20;
  month = dateAndTime + 4;
  day_str = dateAndTime + 0;
  day_num = dateAndTime + 8;
  hour = dateAndTime + 11;
  min = dateAndTime + 14;
  sec = dateAndTime + 17;

  /* compile the string */
  /* CAMPATH/FILETYPEPATH/YYYY-MM-DD_HH_MM_SS/\0 */
  sprintf_s(  filepath,
    FILENAME_LEN,
    "%s%s-%s-%s_%s_%s_%s/\0",
    filetypeBasePath,
    year,
    month,
    day_num,
    hour,
    min,
    sec );

  /* deallocate memory */
  free (dateAndTime);

  return 0;
}

char* Observation::mntSlew( char* value )
{
#ifndef _WACAM
  return invalidCommand( value );
```

```
#endif

  char* retMsg ;        // return message to the caller
  int state;
  enum {
    SLEWING,            // slewing
    OP_IN_PROG,         // operation in progress
    INV
  };

  retMsg = (char *)malloc( RET_MSG_LEN );
  state = INV;          // assume invalid state

  /* c */
  if( !strcmp( value, "TRUE\0" ) )
  {

    /* if a mount operation is not currently in progress... */
    if( !mountOpInProgress )
    {

      /* slew */
      state = SLEWING;
#ifndef _NOHARDWARE
      mountOpInProgress = true;
      startMountOp = true;
      mountOpType = MOUNT_SLEW_TYPE;
#endif
    }
    /* operation in progress */
    else
      state = OP_IN_PROG;

  }

  /* select the appropriate return message */
  switch ( state )
  {
  case SLEWING:
    sprintf_s( retMsg, RET_MSG_LEN, "Slewing mount.\r\n\0" );
    /* update XML file */
    sprintf_s( xmlFields.mntSlewingState, XML_VALUE_LEN, "TRUE\0" );
    mountStatus->printTree( MOUNT_XML_FILE_NAME );
    break;

  case OP_IN_PROG:
    sprintf_s( retMsg, RET_MSG_LEN, "Mount operation already in progress..\r\n\0" );
    break;

  default:
    sprintf_s( retMsg, RET_MSG_LEN, "Invalid command.\r\n\0" );
    break;
  }

  cerr << retMsg;
  /* return the message */
  return retMsg;
}


char* Observation::mntHome( char* value )
{
#ifndef _WACAM
  return invalidCommand( value );
#endif

  /* if an observation is in progress... */
  if ( obsInProgress )
    /* ignore the command */
    return getObsInProgressMsg();
```

```
  char* retMsg ;        // return message to the caller
  int state;
  enum {
    HOMING,             // homing
    OP_IN_PROG,         // operation in progress
    INV
  };

  retMsg = (char *)malloc( RET_MSG_LEN );
  state = INV;          // assume invalid state

  /* c */
  if( !strcmp( value, "TRUE\0" ) )
  {

    /* if a mount operation is not currently in progress... */
    if( !mountOpInProgress )
    {
      /* slew */
      state = HOMING;
#ifndef _NOHARDWARE
      mountOpInProgress = true;
      startMountOp = true;
      mountOpType = MOUNT_HOME_TYPE;
#endif
    }
    /* operation in progress */
    else
      state = OP_IN_PROG;
  }

  /* select the appropriate return message */
  switch ( state )
  {
  case HOMING:
    sprintf_s( retMsg, RET_MSG_LEN, "Homing mount.\r\n\0" );
    /* update XML file */
    sprintf_s( xmlFields.mntHomingState, XML_VALUE_LEN, "TRUE\0" );
    mountStatus->printTree( MOUNT_XML_FILE_NAME );
    break;

  case OP_IN_PROG:
    sprintf_s( retMsg, RET_MSG_LEN, "Mount operation already in progress..\r\n\0" );
    break;

  default:
    sprintf_s( retMsg, RET_MSG_LEN, "Invalid command.\r\n\0" );
    break;
  }

  cerr << retMsg;
  /* return the message */
  return retMsg;
}


char* Observation::mntAbort( char* value )
{
#ifndef _WACAM
  return invalidCommand( value );
#endif

  /* if an observation is in progress... */
  if ( obsInProgress )
    /* ignore the command */
    return getObsInProgressMsg();

  char* retMsg ;        // return message to the caller
  int state;
  enum {
    ABORT,
```

```
    INV
  };

  retMsg = (char *)malloc( RET_MSG_LEN );
  state = INV;          // assume invalid state

  /* c */
  if( !strcmp( value, "TRUE\0" ) )
  {

    /* abort */
    state = ABORT;

#ifndef _NOHARDWARE
    /* this prevents the park+disconnect from "disconnecting" when aborted */
    mountOpType = MOUNT_INVALID_TYPE;
    mnt->abortOperation();
#endif

  }

  /* select the appropriate return message */
  switch ( state )
  {
  case ABORT:
    sprintf_s( retMsg, RET_MSG_LEN, "Mount operation aborted.\r\n\0" );
    break;

  default:
    sprintf_s( retMsg, RET_MSG_LEN, "Invalid command.\r\n\0" );
    break;
  }

  cerr << retMsg;
  /* return the message */
  return retMsg;
}

MountCtrl* Observation::getMount()
{
  return mnt;
}

int Observation::getMountOpType()
{
  return mountOpType;
}

bool Observation::isMountOpInProgress()
{
  return mountOpInProgress;
}

bool Observation::isMountOpStartRequested()
{
  if ( startMountOp )
  {
    startMountOp = false;
    return true;
  }
  else
    return false;
}


int Observation::clearMountOpInProgress()
{
  mountOpInProgress = false;
  /* clear XML slewing states */
  sprintf_s( xmlFields.mntSlewingState, XML_VALUE_LEN, "FALSE\0" );
  sprintf_s( xmlFields.mntHomingState, XML_VALUE_LEN,  "FALSE\0" );
```

```
  /* if we also disconnected... */
  if( mountOpType == MOUNT_PARK_TYPE )
  {
    if( !mnt->testAndClearFailedStatus() )
      sprintf_s( xmlFields.mntConnectedState, XML_VALUE_LEN,  "FALSE\0" );
  }
  else if( mountOpType == MOUNT_CONNECT_TYPE )
  {
    if( !mnt->testAndClearFailedStatus() )
      sprintf_s( xmlFields.mntConnectedState, XML_VALUE_LEN,  "TRUE\0" );
  }
  mountStatus->printTree( MOUNT_XML_FILE_NAME );
  return 0;
}

char* Observation::mntAzi( char* value )
{
  char* retMsg ;        // return message to the caller
  char* currentNumber;  // string containing the current number
  int state;
  enum {
    VALID,              //  - valid azi specified
    INV                 //  - invalid command
  };

  retMsg        = (char *)malloc( RET_MSG_LEN );
  currentNumber = (char *)malloc( RET_MSG_LEN );
  state = INV;          // assume invalid state

  /* delimeter positions */
  char *delimPos1, *delimPos2;
  /* degrees, minutes, seconds */
  int d, m, s;

  /* make sure the string consists of only digits and delimeters */
  char* temp = value;
  bool isValid = true;
  while( *temp != '\0' )
  {
    if( !isdigit(*temp) && ( *temp != TIME_DELIM ) )
      isValid = false;
    temp++;
  }

  if( (delimPos1 = strchr( value, ALT_AZI_DELIM ) ) )
    if( (delimPos2 = strchr( delimPos1+1, ALT_AZI_DELIM ) ) )
    {

      /* attempt to extract the degrees */
      memset( currentNumber, 0, RET_MSG_LEN );
      strncpy_s( currentNumber, RET_MSG_LEN, value, (delimPos1 - value) );
      d = atoi( currentNumber );

      /* attempt to extract the minutes */
      memset( currentNumber, 0, RET_MSG_LEN );
      strncpy_s( currentNumber, RET_MSG_LEN, delimPos1 + 1, (delimPos2 - delimPos1) );
      m = atoi( currentNumber );

      /* attempt to extract the seconds */
      memset( currentNumber, 0, RET_MSG_LEN );
      strncpy_s( currentNumber, RET_MSG_LEN, delimPos2 + 1, RET_MSG_LEN );
      s = atoi( currentNumber );

      /* validate values */
      if( d >= 0 && d <= 359 && isValid)
        if( m >= 0 && m <= 59 )
          if( s >= 0 && s <= 59 )
            state = VALID;

      if( state != VALID )
      {
```

```
        d = -1; m = -1; s = -1;
      }
    }

    /* select the appropriate return message */
    switch ( state )
    {
    case VALID:
      /* fast */
      sprintf_s( retMsg, RET_MSG_LEN, "Azimuth set to %i:%i:%i.\r\n\0", d, m, s );
      azimuth.degrees = d; azimuth.minutes = m; azimuth.seconds = s;
      break;

    default:
      sprintf_s( retMsg, RET_MSG_LEN, "Invalid command.\r\n\0" );
      break;
    }

    cerr << retMsg;
    /* return the message */
    return retMsg;
}


char* Observation::mntAlt( char* value )
{
  char* retMsg ;        // return message to the caller
  char* currentNumber;  // string containing the current number
  int state;
  enum {
    VALID,              //  - valid alt specified
    INV                 //  - invalid command
  };

  retMsg        = (char *)malloc( RET_MSG_LEN );
  currentNumber = (char *)malloc( RET_MSG_LEN );
  state = INV;          // assume invalid state

  /* delimeter positions */
  char *delimPos1, *delimPos2;
  /* degrees, minutes, seconds */
  int d, m, s;

  /* make sure the string consists of only digits and delimeters */
  char* temp = value;
  bool isValid = true;
  while( *temp != '\0' )
  {
    if( !isdigit(*temp) && ( *temp != TIME_DELIM ) )
      isValid = false;
    temp++;
  }

  if( (delimPos1 = strchr( value, ALT_AZI_DELIM ) ) )
    if( (delimPos2 = strchr( delimPos1+1, ALT_AZI_DELIM ) ) )
    {

      /* attempt to extract the degrees */
      memset( currentNumber, 0, RET_MSG_LEN );
      strncpy_s( currentNumber, RET_MSG_LEN, value, (delimPos1 - value) );
      d = atoi( currentNumber );

      /* attempt to extract the minutes */
      memset( currentNumber, 0, RET_MSG_LEN );
      strncpy_s( currentNumber, RET_MSG_LEN, delimPos1 + 1, (delimPos2 - delimPos1) );
      m = atoi( currentNumber );

      /* attempt to extract the seconds */
      memset( currentNumber, 0, RET_MSG_LEN );
      strncpy_s( currentNumber, RET_MSG_LEN, delimPos2 + 1, RET_MSG_LEN );
      s = atoi( currentNumber );
```

```
      /* validate values */
      if( d >= 0 && d <= 359 && isValid)
        if( m >= 0 && m <= 59 )
          if( s >= 0 && s <= 59 )
            state = VALID;

      if( state != VALID )
      {
        d = -1; m = -1; s = -1;
      }

    }

    /* select the appropriate return message */
    switch ( state )
    {
    case VALID:
      /* fast */
      sprintf_s( retMsg, RET_MSG_LEN, "Altitude set to %i:%i:%i.\r\n\0", d, m, s );
      altitude.degrees = d; altitude.minutes = m; altitude.seconds = s;
      break;

    default:
      sprintf_s( retMsg, RET_MSG_LEN, "Invalid command.\r\n\0" );
      break;
    }

    cerr << retMsg;
    /* return the message */
    return retMsg;
}


MntPosition Observation::getAzimuth()
{
  return azimuth;
}


MntPosition Observation::getAltitude()
{
  return altitude;
}

bool Observation::positionTimer()
{
  time_t theTime;
  time( &theTime );

  if ( ( theTime - lastPositionUpdate ) > 4 )
  {
    lastPositionUpdate = (long)theTime;
    return true;
  }
  else
    return false;
}

int Observation::updateXmlPosition( PosUpdate* pos )
{
  if( pos == 0 )
    return 1;
  sprintf_s( xmlFields.mntAltDegState, XML_VALUE_LEN, "%i\0", (*pos).altd );
  sprintf_s( xmlFields.mntAltMinState, XML_VALUE_LEN, "%i\0", (*pos).altm );
  sprintf_s( xmlFields.mntAltSecState, XML_VALUE_LEN, "%2.0f\0", (*pos).alts );
  sprintf_s( xmlFields.mntAziDegState, XML_VALUE_LEN, "%i\0", (*pos).azid );
  sprintf_s( xmlFields.mntAziMinState, XML_VALUE_LEN, "%i\0", (*pos).azim );
  sprintf_s( xmlFields.mntAziSecState, XML_VALUE_LEN, "%2.0f\0", (*pos).azis );
  mountStatus->printTree( MOUNT_XML_FILE_NAME );
  return 0;
```

```
}

int Observation::updateXmlTemperature()
{
  int temp = cam->readTemp();
  if( temp == CAM_INVALID_TEMP )
    return 1;
  else
  {
    sprintf_s( xmlFields.camCcdTempState, XML_VALUE_LEN, "%i\0", temp );
    camStatus->printTree( CAM_XML_FILE_NAME );
    return 0;
  }

}
```

OpticsCtrl.h

```
#ifndef OPTICSCTRL_
#define OPTICSCTRL_
#endif

#include <windows.h>
#include <iostream>
#include <tchar.h>
#include <string.h>
#include <time.h>
#include <assert.h>

using namespace std;

/* defines */
/*************************************************************************/

/* general */
#define MAXDATABYTES      50    // maximum num of bytes that may be txed/rxed
#define NUM_RETS          1     // number of '\r' sent by SMARTMOTOR to end tx
#define HOME_POSITION     1     // home filter position

/* these are used to interpret the messages returned from the SMARTMOTOR */
#define GET_SHUTTER_RET_MSG         "SHTR"  // shutter return message
#define GET_SHUTTER_RET_MSG_OFFSET  5       // location of value in return msg
#define GET_FILTER_RET_MSG          "FILT"  // filter return message
#define GET_FILTER_RET_MSG_OFFSET   5       // location of value in return msg

#define TELECAM_SHUTTER_OPEN  "Open\0"
#define TELECAM_SHUTTER_CLOSE "Closed\0"


/*************************************************************************/
class OpticsCtrl
{
public:

  /* OpticsCtrl()
   *
   *  Class Constructor (initialization).
   *
   * Input:
   *  none
   * Returns:
   *  nothing
   */
  OpticsCtrl();

  /* open()
   *
   *    Opens and configures a connection with specified com port.  If an
   *  incorrect com port is specified (not an integer > 0), COM1 is used
   *  by default.  ** CURRENTLY HARDCODED TO ONLY ACCEPT COM 1-4 **  If
   *  anything above COM4 is requested, COM1 is used by default.
```

```
 *
 * Input:
 *  portNum - integer specifying which com port to use (COM X)
 * Returns:
 *  0 on success, 1 on error
 */
int open(int portNum);

/* getPosition()
 *
 *    Sends command to ask the SMARTMOTOR system for its current filter
 *  wheel position and then waits for a reply.  Information is extracted
 *  from reply and current position is returned to calling function.
 *
 * Input:
 *  none
 * Returns:
 *  integer representing current filter position (1 -> n)
 */
int getPosition();

/* setPosition()
 *
 *  Sends commands to move filter wheel to specified position.  Invalid
 * position values are not checked here as they are handled by the
 * SMARTMOTOR system itself.  Waits for the SMARTMOTOR system to send
 * its confimation.  To confirm positioning getPosition() should be used.
 *
 * Input:
 *  pos - integer specifying which position to move to.
 * Returns:
 *  new position of filter wheel
 */
int setPosition(int pos);

/* setShutter()
 *
 *    Sends commands to change the shutter to specified state.  If the
 *  specified state is anything but open (including invalid values), a closed
 *  state is assumed (for safety).
 *
 * Input:
 *  open - integer specifying which position to place shutter in. 1 for opened,
 *         anything else for closed.
 * Returns:
 *  integer representing current shutter status ( 1 = open, 0 = closed )
 */
int setShutter(int open);

/* getShutter()
 *
 * S  ends commands to ask the SMARTMOTOR system for its current shutter
 * position and then waits for a reply.  Information is extracted from
 * reply and current shutter position is returned to calling function.
 *
 * Input:
 *  none
 * Returns:
 *  integer representing current shutter status ( 1 = open, 0 = closed )
 */
int getShutter();

/* writePort()
 *
 *  Sends given command to SMARTMOTOR system via com connection using
 * using hCom handle.
 *
 * Input:
 *  msg - char ptr to null-terminated message to be sent
 *
 * Return:
```

```
   *  0 on success, 1 on failure
   */
  int writePort(char *msg);

  /* readPort()
   *
   *  Reads data sent from SMARTMOTOR by checking the com input buffer.
   *
   * Input:
   *  msg - char ptr to null-terminated string where message is stored.
   *
   * Return:
   *  0 on success, 1 on failure
   */
  int readPort(char *msg);

  /* close()
   *
   *  Closes the com connection used for communication between the PC
   *  and the SMARTMOTOR system.
   *
   * Input:
   *  none
   * Return:
   *  0 always
   */
  int close();

private:
  bool portRdy;                 // signifies that port is ready (connected)
  HANDLE hCom;                  // handle to com port
  DCB dcb;                      // comm port settings
  DWORD bytesWritten;           // number of bytes transmitted to SMARTMOTOR
  DWORD bytesRead;              // number of bytes read from SMARTMOTOR

};
```

OpticsCtrl.cpp

```
#include "OpticsCtrl.h"


OpticsCtrl::OpticsCtrl()
{
  portRdy = false;
}

int OpticsCtrl::open( int portNum )
{

  /* if we are already connected */
  if( portRdy )
    /* dont bother trying to connect again */
    return 0;

  /* select COM port based on portNum */
  TCHAR* pcCommPort = NULL;
  switch( portNum )
  {
  case 2:
    pcCommPort = TEXT( "COM2" );
    break;

  case 3:
    pcCommPort = TEXT( "COM3" );
    break;

  case 4:
    pcCommPort = TEXT( "COM4" );
    break;
```

```
default:
  /* choose COM1 if an invalid selection is made */
  pcCommPort = TEXT( "COM1" );
}

/* attempt to establish connection */
hCom = CreateFile(
  pcCommPort,                     // port name
  GENERIC_READ | GENERIC_WRITE,   // read and write
  0,                              // exclusive-access (can't share COM port)
  0,                              // default security attributes
  OPEN_EXISTING,                  // must use OPEN_EXISTING
  0,                              // don't use overlapping I/O
  NULL                            // no template file for COM port
  );

/* test connection */
if ( hCom == INVALID_HANDLE_VALUE )
{
  /* invalid connection */
  cerr << "Failed to establish connection.\n";
  return 1;
}

/* set up dcb for loading current comm configuration */
SecureZeroMemory( &dcb, sizeof( DCB ) );
dcb.DCBlength = sizeof( DCB );

/* attempt to get current comm state */
if( !GetCommState( hCom, &dcb ) )
{
  // error
  cerr << "Failed GetCommState.\n";
  return 1;
}

/* current settings loaded, now change settings for 8n1 @ 9600 baud */
dcb.BaudRate  = CBR_9600;     // 9600 baud rate
dcb.ByteSize  = 8;            // xmit/recv data size
dcb.Parity    = NOPARITY;     // no parity
dcb.StopBits  = ONESTOPBIT;   // one stop bit

/* attempt to load new settings */
if( !SetCommState( hCom, &dcb ) )
{
  // error!
  cerr << "Failed SetCommState.\n";
  return 1;
}

/* This is not required, address of motor is set to #1 by default startup */
///* set the smartmotors address by sending it via comm link */
///* 129 = motor #1 (see smartmotor reference) */
//sprintf_s( msg, sizeof(msg),"\129" );
//
///* attempt to write to port */
//if( !WriteFile(
//  hCom,                   // handle to comm port
//  msg,                    // buffer containing message
//  sizeof(msg),            // number of bytes to write
//  &bytesWritten,          // number of bytes written (set by WriteFile)
//  NULL                    // not used
//  )
//  )
//{
//  cerr << "Couldn't send address to motor.\n";
//  return 1;
//}

/* set up the timeout settings for the connection */
```

```
  COMMTIMEOUTS timeouts;
  timeouts.ReadIntervalTimeout          = MAXDWORD; // max timeout
  timeouts.ReadTotalTimeoutMultiplier   = 0;
  timeouts.ReadTotalTimeoutConstant     = 0;
  timeouts.WriteTotalTimeoutMultiplier  = 0;
  timeouts.WriteTotalTimeoutConstant    = 0;

  if (!SetCommTimeouts(hCom, &timeouts))
  {
    cerr << "Error setting timeouts.\n";
    return 1;
  }

  /* everything was successful, set port status accordingly */
  portRdy = true;

  /* home the filter and close the shutter */
  setPosition( HOME_POSITION );
  setShutter( 0 );

  return 0;
}

int OpticsCtrl::setPosition( int pos )
{
  /* if we aren't connected... */
  if( !portRdy )
    /* don't try to move the filter wheel */
    return -1;

  /* buffer for sending commands */
  char msg[MAXDATABYTES];
  /* buffer for receiving confirmation */
  char buf[MAXDATABYTES];
  /* information to be returned */
  int retVal = -1;

  /* send appropiate commands */
  /* g=n\r  ...   GOSUB4\r */
  sprintf_s( msg, sizeof( msg ),"g=%i\r", pos );
  writePort( msg );
  sprintf_s( msg, sizeof(msg),"GOSUB4\r" );
  writePort( msg );

  /* prepare to read output from SMARTMOTOR */
  int i = 0;              // index of local buffer
  int j = 0;              // number of '\r' recvd
  int k = 0;              // index of recvd buf
  int n_rets = NUM_RETS;  // number of '\r' to wait for

  /* while we still haven't recvd the max num of '\r'... */
  while ( ( j < n_rets ) )
  {
    /* read any characters recvd from SMARTMOTOR */
    readPort( msg );

    /* for each byte read... */
    while ( bytesRead > 0 )
    {
      /* copy char into buffer, if char is '\r'... */
      if ( ( buf[i++] = msg[k++] ) == '\r' )
      {
        /* increment num of '\r' recvd */
        j++;
      }
      /* decrement num of bytes (chars) remaining */
      bytesRead--;
    }
    /* reset recvd char string index */
    k=0;
  }
```

```
  /* terminate the recvd character script */
  buf[i-1] = 0;

  /* if the correct message was recvd... */
  if ( strstr( buf, GET_FILTER_RET_MSG ) )
  {
    /* Both cam optics return SHTR:X where X is 0 (close) or 1 (open) */
    /* extract the return information (convert from ascii to int) */
    retVal = buf[GET_FILTER_RET_MSG_OFFSET] - 0x30;
    return retVal;
  }
  /* if not... */
  else
  {
    /* error message */
    cerr << "Invalid return value.\n";
  }

  return retVal;
}

int OpticsCtrl::getPosition()
{
  /* if we aren't connected... */
  if( !portRdy )
    /* don't try to get the filter wheel position */
    return -1;

  /* buffer for sending commands */
  char msg[MAXDATABYTES];
  /* buffer for receiving confirmation */
  char buf[MAXDATABYTES];
  /* get position value */
  int pos = -1;

  /* send appropriate commands to SMARTMOTOR */
  /* g=-1\r  ...  GOSUB4\r */
  sprintf_s( msg, sizeof( msg ),"g=%i\r", pos );
  writePort( msg );
  sprintf_s( msg, sizeof(msg),"GOSUB4\r", 1);
  writePort( msg );

  /* read the buffer */
  int i = 0;              // index of local buffer
  int j = 0;              // numer of '\r' recvd
  int k = 0;              // index of recvd buf
  int n_rets = NUM_RETS;  // number of '\r' to wait for

  /* while we still haven't recvd the max num of '\r'... */
  while ( ( j < n_rets ) )
  {
    /* read any characters recvd from SMARTMOTOR */
    readPort( msg );

    /* for each byte read... */
    while ( bytesRead > 0 )
    {
      /* copy char into buffer, if char is '\r'... */
      if ( ( buf[i++] = msg[k++] ) == '\r' )
      {
        /* increment num of '\r' recvd */
        j++;
      }
      /* decrement num of bytes (chars) remaining */
      bytesRead--;
    }
    /* reset recvd char string index */
    k=0;
  }
  /* terminate the recvd character script */
```

```
  buf[i-1] = 0;

  /* information to be returned */
  int retVal = 0;

  /* if the correct message was recvd... */
  if ( strstr( buf, GET_FILTER_RET_MSG ) )
  {
    /* extract the return information (convert from ascii to int) */
    retVal = buf[GET_FILTER_RET_MSG_OFFSET] - 0x30;
  }
  /* if not... */
  else
  {
    /* error message */
    cerr << "Invalid return value.\n";
  }

  return retVal;
}


int OpticsCtrl::setShutter( int open )
{
  /* if we aren't connected... */
  if( !portRdy )
    /* don't try to set the shutter */
    return -1;

  /* buffer for sending commands */
  char msg[MAXDATABYTES];
  /* position of filter */
  int pos;

  /* Load the correct position into the shutter variable */
  if( open )
    pos = 1;
  else
    pos = 0;

  /* send appropiate commands */
  sprintf_s( msg, sizeof( msg ),"d=%i\r", pos );
  writePort( msg );
  sprintf_s( msg, sizeof(msg),"GOSUB1\r", 1);
  writePort( msg );

#ifdef _WACAM
  /* buffer for receiving commands */
  char buf[MAXDATABYTES];

  /* prepare to read output from SMARTMOTOR */
  int i = 0;              // index of local buffer
  int j = 0;              // numer of '\r' recvd
  int k = 0;              // index of recvd buf
  int n_rets = NUM_RETS;  // number of '\r' to wait for

  /* while we still haven't recvd the max num of '\r'... */
  while ( ( j < n_rets ) )
  {
    /* read any characters recvd from SMARTMOTOR */
    readPort( msg );

    /* for each byte read... */
    while ( bytesRead > 0 )
    {
      /* copy char into buffer, if char is '\r'... */
      if ( ( buf[i++] = msg[k++] ) == '\r' )
      {
        /* increment num of '\r' recvd */
        j++;
      }
```

```
      /* decrement num of bytes (chars) remaining */
      bytesRead--;
    }
    /* reset recvd char string index */
    k=0;
  }

  /* terminate the recvd character script */
  buf[i-1] = 0;


  /* information to be returned */
  int retVal = 0;

  /* if the correct message was recvd... */
  if ( strstr( buf, GET_SHUTTER_RET_MSG ) )
  {
    /* The wide cam optics return SHTR:X where X is 0 (close) or 1 (open) */
    /* extract the return information (convert from ascii to int) */
    retVal = buf[GET_FILTER_RET_MSG_OFFSET] - 0x30;
    return retVal;
  }
  /* if not... */
  else
  {
    /* error message */
    cerr << "Invalid return value.\n";
  }

  return retVal;
#else
  return pos;
#endif


}

int OpticsCtrl::getShutter()
{
  /* if we aren't connected... */
  if( !portRdy )
    /* don't try to get the shutter position */
    return -1;

  /* buffer for sending commands */
  char msg[MAXDATABYTES];
  /* buffer for receiving confirmation */
  char buf[MAXDATABYTES];
  /* get shutter value */
  int pos = -1;

  /* send appropriate commands to SMARTMOTOR */
  sprintf_s( msg, sizeof( msg ),"d=%i\r", pos );
  writePort( msg );

  sprintf_s( msg, sizeof(msg),"GOSUB1\r", 1);
  writePort( msg );

  /* read the buffer */
  int i = 0;              // index of local buffer
  int j = 0;              // numer of '\r' recvd
  int k = 0;              // index of recvd buf
  int n_rets = NUM_RETS;  // number of '\r' to wait for

  /* while we still haven't recvd the max num of '\r'... */
  while ( ( j < n_rets ) )
  {
    /* read any characters recvd from SMARTMOTOR */
    readPort( msg );

    /* for each byte read... */
```

```
    while ( bytesRead > 0 )
    {
      /* copy char into buffer, if char is '\r'... */
      if ( ( buf[i++] = msg[k++] ) == '\r' )
      {
        /* increment num of '\r' recvd */
        j++;
      }
      /* decrement num of bytes (chars) remaining */
      bytesRead--;
    }
    /* reset recvd char string index */
    k=0;
  }
  /* terminate the recvd character script */
  buf[i-1] = 0;

  /* information to be returned */
  int retVal = 0;

  /* if the correct message was recvd... */
  if ( strstr( buf, GET_SHUTTER_RET_MSG ) )
  {
#ifdef _WACAM

    /* extract the return information (convert from ascii to int) */
    retVal = buf[GET_FILTER_RET_MSG_OFFSET] - 0x30;
    return retVal;

#else
    /* The narrow cam optics return SHTR Open or SHTR Close */

    /* test for close */
    if( strstr( buf, TELECAM_SHUTTER_CLOSE ) )
      return 0;
    else if( strstr( buf, TELECAM_SHUTTER_OPEN ) )
      return 1;
    else
      return -1;

#endif
  }
  /* if not... */
  else
  {
    /* error message */
    cerr << "Invalid return value.\n";
  }

  return retVal;

}
int OpticsCtrl::writePort( char *msg )
{
  /* If the port is ready for communication... */
  if( portRdy )
  {
    if( !WriteFile( hCom, msg, (DWORD)strlen(msg), &bytesWritten, 0 ) )
    {
      cerr << "Write failed.\n";
      return 0;
    }
  }
  /* Port not ready, return fail status */
  else
  {
    cerr << "Write failed, port not ready.\n";
    return 0;
  }

  /* Write sucessful, return success status */
```

```
  return 1;
} // done

int OpticsCtrl::readPort( char *msg )
{
  /* If the port is ready for communication... */
  if( portRdy )
  {
    if( !ReadFile( hCom, msg, 20, &bytesRead, 0 )  )
    {
      cout << "Read failed: " << GetLastError() <<"\n";
      return 0;
    }
  }
  /* Port not ready, return fail status */
  else
  {
    cerr << "Read failed, port not ready.\n";
    return 0;
  }
  return 1;
}
int OpticsCtrl::close( )
{
  /* if we aren't connected... */
  if( !portRdy )
    /* don't bother trying to disconnect */
    return 1;
  else
    /* disconnect */
    if( hCom != INVALID_HANDLE_VALUE )
    {
      CloseHandle( hCom );
      hCom = INVALID_HANDLE_VALUE;
      portRdy = false;
    }
    return 0;
}
```

TcpIpServer.h

```
#ifndef TCPIPSERVER_H
#define TCPIPSERVER_H


#include <windows.h>
#include <iostream>
using namespace std;

/* This will link to wsock32.lib */
#pragma comment( lib, "WSock32.Lib" )

/* defines */
#define MAX_NUM_CLIENTS 1       // max num of clients serviced simultaneously

/* error message indicies */
#define E_NO_ERROR                          0
#define E_COULDNT_OBTAIN_HOST_INFO          1
#define E_COULDNT_INIT_WINSOCK              2
#define E_COULDNT_CREATE_SERVER_SOCKET      3
#define E_COULDNT_BIND_SERV_SOCKET_TO_PORT  4
#define E_UNABLE_TO_LISTEN                  5
#define E_CANT_RECV_NOT_CONNECTED           6
#define E_CANT_SEND_NOT_CONNECTED           7


class TcpIpServer
{
public:

  TcpIpServer(int port);
```

```
/**
* A constructor.  Initialies class member variables.
*
* INPUT:
*   port  - integer representing which port server will listen on
* RETURNS
*   nothing
*/

int initServerAndListen();
/**
*   Initializes server socket and begins listening for client connection
* requests.
*
* INPUT:
*   none
* RETURNS
*   0 in success, error code on failure.
*/

int acceptClient();
/**
*   Checks for client awaiting connections.  If client request is found,
* that client is accepted and a socket is created for communication with
* client.
*
* INPUT:
*   none
* OUTPUT:
*   0 on success, error code on failure
*/

int recvFromClient(char *buf, int bufLen, char* delim);
/**
*   Receives data sent from accepted client.  This function polls the
* TCP/IP buffer until either one of the specified delimeters is reached
* or the buffer being written into is full.
*
* INPUT:
*   buf - pointer to buffer which received data is written to
*   bufLen - length of buffer being written to
*   delim - cstring containing all delimiters  ex: ";\r\n"
* RETURNS:
*   number of bytes read (>= 0), or error code (< 0)
*/

int sendToClient(char *buf, int bufLen);
/**
*   Sends data to accepted client.  Specified number of bytes in
* specified buffer are transmitted.
*
* INPUT:
*   buf - pointer to buffer containing data being transmitter
*   bufLen - number of bytes to transmit
* RETURNS:
*   0 on success, error code on failure
*/

int closeClientConn();
/**
*   Closes connection with accepted client.  Because only one client
* is supported by this class (one at a time), this function should be called
* right before attempting to listen for another client request.
*
* INPUT:
*   none
* OUTPUT:
*   always 0
*/

const char* getLastError();
```

```
  /**
  *   Returns cstring pointer to last error message.
  *
  * INPUT:
  *   none
  * OUTPUT:
  *   pointer to cstring containing last error message.
  */


private:

  bool mIsConnected;    /* true when connected to client, false when not */
  int mPort;            /* port on server side used for communcation */
  SOCKET mSSock;        /* server socket, used for listening for clients */
  SOCKET mCSock;        /* client socket, used to rx/tx data to client */
  int mErrorCode;       /* error code of pointing to index of last error msg */
  static const char* mErrorMsg[]; /* char array of error messages */
};



#endif
```

TcpIpServer.cpp

```
#include "TcpIpServer.h"


const char* TcpIpServer::mErrorMsg[] =
{
  "No error message.\0",
  "Couldn't obtain host address information.\0",
  "Couldn't initialize WinSock.\0",
  "Couldn't create the server socket.\0",
  "Couldn't bind server socket to port.\0",
  "Unable to listen.\0",
  "Can't receive data, not connected.\0",
  "Can't send data, not connected.\0"
};


TcpIpServer::TcpIpServer(int port)
{
  mSSock = INVALID_SOCKET;    /* sockets not yet initialized */
  mCSock = INVALID_SOCKET;
  mIsConnected = false;       /* not ready to tx/rx */
  mPort = port;
}


int TcpIpServer::initServerAndListen()
{
  /* declare variables */
  char myName[256];             // our host name
  struct sockaddr_in servAddr;  // our address
  struct hostent *hostInfo;     // host information structure
  WSADATA info;                 // windows socket information data structure


  /**************************************************************************
  *   There are 5 steps used here to initialize the server socket and
  * begin listening for client connectin requests, listed below:
  *
  * 1. Initialize WINSOCK (v1.1 in our case)
  * 2. Obtain our host information
  * 3. Create the server socket
  * 4. Bind the socket to given port
  * 5. Begin listening for client requests
  **************************************************************************/
```

```
  /* 1. Attempt to initialize winsock v1.1 ... */

  if ( WSAStartup( MAKEWORD( 1,1 ), &info ) != 0 )
  {
    /* unsuccessful */
    mErrorCode = E_COULDNT_INIT_WINSOCK;
    return 1;
  }



  /* 2. Obtain our host information */

  /* initialize memory for host info struct */
  memset( &servAddr, 0, sizeof( struct sockaddr_in ) );
  /* get our host name */
  gethostname(myName, sizeof(myName));

  /* attempt get our address info */
  if ( ( hostInfo = gethostbyname( myName ) ) == NULL)
  {
    /* unsuccessful */
    mErrorCode = E_COULDNT_OBTAIN_HOST_INFO;
    return 1;
  }

  /* extract host ip from host info struct */
  servAddr.sin_family = hostInfo->h_addrtype;
  /* set port number */
  servAddr.sin_port = htons(mPort);



  /* 3. Attempt to create the socket */

  if( ( mSSock = socket( AF_INET, SOCK_STREAM, 0 ) ) == INVALID_SOCKET )
  {
    /* unsuccessful */
    mErrorCode = E_COULDNT_CREATE_SERVER_SOCKET;
    return 1;
  }


  /* 4. Attempt to bind the socket to the internet address... */

  if (  bind(  mSSock, ( struct sockaddr *)&servAddr, sizeof( struct sockaddr_in ))
    == SOCKET_ERROR )
  {
    /* unsuccessful */
    closesocket(mSSock);
    mErrorCode = E_COULDNT_BIND_SERV_SOCKET_TO_PORT;
    return 1;
  }


  /* 5. Attempt to listen for clients */

  if ( listen( mSSock, 3 ) != 0 )
  {
    mErrorCode = E_UNABLE_TO_LISTEN;
    return 1;
  }

  return 0;
}


int TcpIpServer::acceptClient()
{
```

```
  /* attempt to accept client connection */
  if ( ( mCSock = accept( mSSock, NULL, NULL ) ) == INVALID_SOCKET )
  {
    /* unsuccessful */
    /* we are not connected */
    mIsConnected = false;
    return 1;
  }
  else
  {
    /* successful */
    /* we are connected */
    mIsConnected = true;
    return 0;
  }

  return 0;
}


int TcpIpServer::recvFromClient(char *buf, int bufLen, char* delim)
{
  /* abort if we are not connected */
  if( !mIsConnected )
  {
    mErrorCode = E_CANT_RECV_NOT_CONNECTED;
    return 1;
  }

  /* declare variables */
  int bcount;   /* total bytes read */
  int br;       /* bytes read this pass */
  bcount = 0; br = 0;

  /* while the buffer isn't full... */
  while ( bcount < ( bufLen - 1 ) )
  {
    /* read a byte from the buffer */
    if ( ( br = recv( mCSock, buf, 1, 0 ) ) > 0 )
    {
      /* increment byte counter */
      bcount += br;

      /* test for EOF */
      if ( *buf == EOF )
      {
        /* EOF found, return */
        return EOF;
      }
      /* then test for delimeter */
      else if ( *buf == *delim )
      {
        /* delimeter found, append \0 and return # bytes read */
        *(buf+1) = '\0';
        return bcount;
      }

      /* neither EOF nor delimeter, get next byte */

      /* move buffer ptr for next read */
      buf += br;
    }
    /* signal an error to the caller */
    else
    {
      return -1;
    }
  }

  return bcount;
}
```

```
int TcpIpServer::sendToClient(char *buf, int bufLen)
{
  /* abort if we are not connected */
  if( !mIsConnected )
  {
    mErrorCode = E_CANT_SEND_NOT_CONNECTED;
    return 1;
  }

  /* declare variables */
  int bcount;
  int br; /* bytes sent this pass */
  bcount = 0; br = 0;

  while (bcount < bufLen)
  { /* loop until full buffer */
    if ( ( br = send( mCSock, buf, bufLen - bcount, 0 ) ) > 0 )
    {
      bcount += br; /* increment byte counter */

      buf += br; /* move buffer ptr for next transmission */
    }

    else if (br < 0) /* signal an error to the caller */
    {
      return 1;
    }
  }
  return bcount;
}


int TcpIpServer::closeClientConn()
{
  /* disconnect from client */
  mIsConnected = false;
  closesocket(mCSock);
  return 0;
}

const char* TcpIpServer::getLastError()
{
  return mErrorMsg[mErrorCode];
}
```

XmlNode.h

```
#ifndef XMLNODE_H
#define XMLNODE_H

#include <windows.h>
#include <fstream>
#include <time.h>

using namespace std;


#define STREAM_BUF_LEN  256
#define INDENT_BUF_LEN  16
#define XML_VALUE_LEN   128

struct xmlFields {

  /* mountStatus.xml */
  char* mntStatus;
  /**/char* mntUpdated;
  /**/char* mntConnectedState;
  /**/char* mntSlewingState;
  /**/char* mntHomingState;
```

```
  /**/char* mntPositionState;
  /******/char* mntAziDegState;
  /******/char* mntAziMinState;
  /******/char* mntAziSecState;
  /******/char* mntAltDegState;
  /******/char* mntAltMinState;
  /******/char* mntAltSecState;

  /* telecamStatus.xml */
  char* camStatus;
  /**/char* camUpdate;
  /**/char* camObs;
  /******/char* camInProgress;
  /******/char* camLastPicName;
  /******/char* camLastPicPath;
  /**/char* camHardware;
  /******/char* camOptics;
  /**********/char* camOptConnectedState;
  /**********/char* camShutterState;
  /**********/char* camFilterState;
  /******/char* camCamera;
  /**********/char* camConnectedState;
  /**********/char* camCcdTempState;

  /* wacamStatus.xml */
  char* wacamStatus;
  /**/char* wacamUpdate;
  /**/char* wacamObs;
  /******/char* wacamInProgress;
  /******/char* wacamLastPicName;
  /******/char* wacamLastPicTime;
  /**/char* wacamHardware;
  /******/char* wacamOptics;
  /**********/char* wacamOptConnectedState;
  /**********/char* wacamShutterState;
  /**********/char* wacamFilterState;
  /******/char* wacamCamera;
  /**********/char* wacamCamConnectedState;
  /**********/char* wacamCcdTempState;
};
typedef struct xmlFields XmlFields;


class XmlNode
{
public:

  XmlNode( char* tag, char** value );
  /**
  *   A constructor.  Consumes a tag name and pointer to a value and creates
  * an XML tag node with given properties.
  *
  * INPUT:
  *   tag - cstring containing tag name
  *   value - pointer to cstring containing value
  */

  int addChild( XmlNode* newChildNode );
  /**
  *   Consumes a pointer to an XmlNode and makes that node a child of the
  * current node.
  *
  * INPUT:
  *   newChildNode - new node to be added as a child
  * RETURNS:
  *   0 on success, 1 on failure
  */

  char* getXmlTag();
  /**
  * Accessor.  Returns cstring ptr to xml tag name.
```

```
  *
  * INPUT:
  *   none
  * OUTPUT:
  *   Ptr to xml tag name (cstring).
  */

  char* getXmlData();
  /**
  * Accessor.  Returns cstring ptr to xml tag value.
  *
  * INPUT:
  *   none
  * OUTPUT:
  *   Ptr to xml tag value (cstring).
  */

  int getNumChildren();
  /**
  * Accessor.  Returns number of children of current node.
  *
  * INPUT:
  *   none
  * OUTPUT:
  *   int representing the number of children
  */

  bool hasChildren();
  /**
  * Accessor.  Checks if the current node has children.
  *
  * INPUT:
  *   none
  * OUTPUT:
  *   true if it does have children, false if not.
  */

  XmlNode** getChildren();
  /**
  * Accessor.  Returns ptr to array of children.
  *
  * INPUT:
  *   none
  * OUTPUT:
  *   Ptr to array of children.
  */

  int initializeMntTree( XmlFields& fieldStruct );
  int initializeCamTree( XmlFields& fieldStruct );
  int initializeWecamTree( XmlFields& fieldStruct );

  char* getUpdateTimePtr();

  int printTree( const char* filename );
  /**
  *   Iterates through XML tree with head node this and prints the tree
  * to a file with the name filename.
  *
  * INPUT:
  *   filename - name of the file to be written
  * OUTPUT:
  *   0 on success, 1 on failure
  */
  int printTreeHelper(
    fstream*  xmlFile,
    XmlNode*  currentNode,
    int       indentationLevel
    );
  int killChildren();

private:
```

```
  char*     myXmlTag;       // xml tag of this node
  char**    myXmlData;      // xml data
  XmlNode** myChildren;     // array of children ptrs
  int       myNumChildren;  // number of children

  char*     writeBuffer;    // buffer used for writing to file
  char*     indentBuffer;   // buffer used for storing indentation
  char*     updateTime;     // last update time
};


#endif
```

XmlNode.cpp

```
#include "XmlNode.h"



XmlNode::XmlNode( char* tag, char** value )
{
  myXmlTag = tag;
  myXmlData = value;
  myChildren = NULL;
  myNumChildren = 0;

  writeBuffer = (char *)malloc( STREAM_BUF_LEN );
  updateTime = (char *)malloc( 128 );
  sprintf_s( updateTime, 128, "unknown\0" );
}


int XmlNode::addChild( XmlNode* newChildNode )
{
  /* if we don't have any children yet... */
  if ( myChildren == NULL )
  {
    /* ...adopt this as our only child one */
    myChildren = (XmlNode**)malloc( sizeof(XmlNode*) );
    *myChildren = newChildNode;
    myNumChildren = 1;
  }
  /* if we do have children... */
  else
  {
    /* find a new place to house our children */
    XmlNode** temp = (XmlNode **)malloc( sizeof(XmlNode*) * ( myNumChildren + 1 ) );

    /* move all of our old children to the new place */
    memcpy( (void*)(temp), (void*)(myChildren), sizeof(XmlNode*) * myNumChildren );

    /* get rid of our old place */
    free( myChildren );
    myChildren = temp;

    /* bring in the new child */
    myChildren[myNumChildren] = newChildNode;
    myNumChildren++;
  }

  return 0;
}


char* XmlNode::getXmlTag()
{
  return myXmlTag;
}
```

```
char* XmlNode::getXmlData()
{
  return *myXmlData;
}


int XmlNode::getNumChildren()
{
  return myNumChildren;
}


bool XmlNode::hasChildren()
{
  if( myChildren == NULL )
    return false;
  else
    return true;
}

XmlNode** XmlNode::getChildren()
{
  return myChildren;
}


int XmlNode::printTree( const char* filename )
{
  /* get the time of the update */
  time_t currentTime;
  time( &currentTime );
  ctime_s( this->updateTime, 128, &currentTime );
  /* get rid of the newline */
  char* temp = updateTime;
  while( *temp != '\0' )
  {
    if( *temp == '\n' )
      *temp = '\0';
    temp++;
  }

  /* decl */
  fstream xmlFile;

  xmlFile.open( filename, ios_base::out );

  /* go! */
  printTreeHelper( &xmlFile, this, 0 );

  /* stop! */
  xmlFile.close();

  return 0;
}


int XmlNode::printTreeHelper(
                            fstream*  xmlFile,
                            XmlNode*  currentNode,
                            int       indentationLevel
                            )
{
  int i;

  /* pre-indent the string */
  for( i = 0; i < indentationLevel; i++ )
    *( writeBuffer + i ) = '\t';
  *(writeBuffer + i) = '\0';

  /* copy the tag data to it */
  sprintf_s(
```

```
    writeBuffer,
    STREAM_BUF_LEN,
    "%s<%s>",
    writeBuffer,
    currentNode->getXmlTag()
    );

  /* print the data to file */
  (*xmlFile).write( writeBuffer, (streamsize)strlen(writeBuffer) );

  /* if the current node has no children... */
  if( !currentNode->hasChildren() )
  {
    /* print its XML data and closing tag */
    sprintf_s(
      writeBuffer,
      STREAM_BUF_LEN,
      "%s</%s>\n",
      currentNode->getXmlData(),
      currentNode->getXmlTag()
      );

    (*xmlFile).write( writeBuffer, (streamsize)strlen(writeBuffer) );
  }
  else
  {
    sprintf_s( writeBuffer, STREAM_BUF_LEN, "\n" );
    (*xmlFile).write( writeBuffer, (streamsize)strlen(writeBuffer) );

    /* visit each child */
    int i;
    for( i = 0; i < currentNode->getNumChildren(); i++ )
      printTreeHelper( xmlFile, currentNode->getChildren()[i], indentationLevel + 1 );
    /* pre-indented the string */
    for( i = 0; i < indentationLevel; i++ )
      *( writeBuffer + i ) = '\t';
    writeBuffer[i + 1] = '\0';



    /* pre-indent the string */
    for( i = 0; i < indentationLevel; i++ )
      *( writeBuffer + i ) = '\t';
    *(writeBuffer + i) = '\0';
    (*xmlFile).write( writeBuffer, (streamsize)strlen(writeBuffer) );

    /* print XML closing tag */
    sprintf_s(
      writeBuffer,
      STREAM_BUF_LEN,
      "</%s>\n",
      currentNode->getXmlTag()
      );
    (*xmlFile).write( writeBuffer, (streamsize)strlen(writeBuffer) );
  }

  return 0;
}


int XmlNode::initializeMntTree(XmlFields& fieldStruct)
{
  /* allocate memory for the fields */
  fieldStruct.mntStatus               = NULL;
  /**/fieldStruct.mntUpdated          = this->updateTime;
  /**/fieldStruct.mntConnectedState   = (char *)malloc( XML_VALUE_LEN );
  /**/fieldStruct.mntSlewingState     = (char *)malloc( XML_VALUE_LEN );
  /**/fieldStruct.mntHomingState      = (char *)malloc( XML_VALUE_LEN );
  /**/fieldStruct.mntPositionState    = NULL;
  /******/fieldStruct.mntAziDegState  = (char *)malloc( XML_VALUE_LEN );
  /******/fieldStruct.mntAziMinState  = (char *)malloc( XML_VALUE_LEN );
```

```
  /******/fieldStruct.mntAziSecState  = (char *)malloc( XML_VALUE_LEN );
  /******/fieldStruct.mntAltDegState  = (char *)malloc( XML_VALUE_LEN );
  /******/fieldStruct.mntAltMinState  = (char *)malloc( XML_VALUE_LEN );
  /******/fieldStruct.mntAltSecState  = (char *)malloc( XML_VALUE_LEN );

  /* initialize the items */
  //strcpy_s( fieldStruct.mntUpdated, XML_VALUE_LEN, "INV\0" );
  strcpy_s( fieldStruct.mntConnectedState, XML_VALUE_LEN, "INV\0" );
  strcpy_s( fieldStruct.mntSlewingState, XML_VALUE_LEN, "INV\0" );
  strcpy_s( fieldStruct.mntHomingState, XML_VALUE_LEN, "INV\0" );
  strcpy_s( fieldStruct.mntAziDegState, XML_VALUE_LEN, "INV\0" );
  strcpy_s( fieldStruct.mntAziMinState, XML_VALUE_LEN, "INV\0" );
  strcpy_s( fieldStruct.mntAziSecState, XML_VALUE_LEN, "INV\0" );
  strcpy_s( fieldStruct.mntAltDegState, XML_VALUE_LEN, "INV\0" );
  strcpy_s( fieldStruct.mntAltMinState, XML_VALUE_LEN, "INV\0" );
  strcpy_s( fieldStruct.mntAltSecState, XML_VALUE_LEN, "INV\0" );

  /* create all required sub-nodes */
  XmlNode* mntUpdated         = new XmlNode( "updated\0", &(fieldStruct.mntUpdated) );
  XmlNode* mntConnectedState  = new XmlNode( "connectedState\0",
&(fieldStruct.mntConnectedState) );
  XmlNode* mntSlewingState    = new XmlNode( "slewingState\0", &(fieldStruct.mntSlewingState) );
  XmlNode* mntHomingState     = new XmlNode( "homingState\0", &(fieldStruct.mntHomingState) );
  XmlNode* mntPositionState   = new XmlNode( "positionState\0", &(fieldStruct.mntPositionState) );
  XmlNode* mntAziDegState     = new XmlNode( "aziDegState\0", &(fieldStruct.mntAziDegState) );
  XmlNode* mntAziMinState     = new XmlNode( "aziMinState\0", &(fieldStruct.mntAziMinState) );
  XmlNode* mntAziSecState     = new XmlNode( "aziSecState\0", &(fieldStruct.mntAziSecState) );
  XmlNode* mntAltDegState     = new XmlNode( "altDegState\0", &(fieldStruct.mntAltDegState) );
  XmlNode* mntAltMinState     = new XmlNode( "altMinState\0", &(fieldStruct.mntAltMinState) );
  XmlNode* mntAltSecState     = new XmlNode( "altSecState\0", &(fieldStruct.mntAltSecState) );


  /* implement hierarchy */
  mntPositionState->addChild( mntAziDegState );
  mntPositionState->addChild( mntAziMinState );
  mntPositionState->addChild( mntAziSecState );
  mntPositionState->addChild( mntAltDegState );
  mntPositionState->addChild( mntAltMinState );
  mntPositionState->addChild( mntAltSecState );
  this->addChild( mntUpdated );
  this->addChild( mntConnectedState );
  this->addChild( mntSlewingState );
  this->addChild( mntHomingState );
  this->addChild( mntPositionState );

  return 0;
}


int XmlNode::initializeCamTree( XmlFields& fieldStruct )
{

  /* camStatus.xml */
  fieldStruct.camStatus                       = NULL;
  /**/fieldStruct.camUpdate                   = this->updateTime;
  /**/fieldStruct.camObs                      = NULL;
  /******/fieldStruct.camInProgress           = (char*)malloc( XML_VALUE_LEN );
  /******/fieldStruct.camLastPicName          = (char*)malloc( XML_VALUE_LEN );
  /******/fieldStruct.camLastPicPath          = (char*)malloc( XML_VALUE_LEN );
  /**/fieldStruct.camHardware                 = NULL;
  /******/fieldStruct.camOptics                = NULL;
  /**********/fieldStruct.camOptConnectedState = (char*)malloc( XML_VALUE_LEN );
  /**********/fieldStruct.camShutterState     = (char*)malloc( XML_VALUE_LEN );
  /**********/fieldStruct.camFilterState      = (char*)malloc( XML_VALUE_LEN );
  /******/fieldStruct.camCamera               = NULL;
  /**********/fieldStruct.camConnectedState   = (char*)malloc( XML_VALUE_LEN );
  /**********/fieldStruct.camCcdTempState     = (char*)malloc( XML_VALUE_LEN );

  //strcpy_s( fieldStruct.camUpdate,           XML_VALUE_LEN, "?\0" );
  strcpy_s( fieldStruct.camInProgress,       XML_VALUE_LEN, "FALSE\0" );
  strcpy_s( fieldStruct.camLastPicName,      XML_VALUE_LEN, "default_image.png\0" );
```

```
  strcpy_s( fieldStruct.camLastPicPath,       XML_VALUE_LEN, "./\0" );
  strcpy_s( fieldStruct.camOptConnectedState, XML_VALUE_LEN, "FALSE\0" );
  strcpy_s( fieldStruct.camShutterState,      XML_VALUE_LEN, "OPEN\0" );
  strcpy_s( fieldStruct.camFilterState,       XML_VALUE_LEN, "1\0" );
  strcpy_s( fieldStruct.camConnectedState,    XML_VALUE_LEN, "FALSE\0" );
  strcpy_s( fieldStruct.camCcdTempState,      XML_VALUE_LEN, "?\0" );

  /* create all required sub-nodes */
  XmlNode* camUpdated           = new XmlNode( "updated\0", &(fieldStruct.camUpdate) );
  XmlNode* camObs               = new XmlNode( "obs\0", &(fieldStruct.camObs) );
  XmlNode* camInProgress        = new XmlNode( "inProgress\0", &(fieldStruct.camInProgress) );
  XmlNode* camLastPicName       = new XmlNode( "lastPicName\0", &(fieldStruct.camLastPicName) );
  XmlNode* camLastPicPath       = new XmlNode( "lastPicPath\0", &(fieldStruct.camLastPicPath) );
  XmlNode* camHardware          = new XmlNode( "hardware\0", &(fieldStruct.camHardware) );
  XmlNode* camOptics            = new XmlNode( "optics\0", &(fieldStruct.camOptics) );
  XmlNode* camOptConnectedState = new XmlNode( "connectedState\0",
&(fieldStruct.camOptConnectedState) );
  XmlNode* camShutterState      = new XmlNode( "shutterState\0", &(fieldStruct.camShutterState) );
  XmlNode* camFilterState       = new XmlNode( "filterState\0", &(fieldStruct.camFilterState) );
  XmlNode* camCamera            = new XmlNode( "camera\0", &(fieldStruct.camCamera) );
  XmlNode* camConnectedState = new XmlNode( "connectedState\0",
&(fieldStruct.camConnectedState) );
  XmlNode* camCcdTempState      = new XmlNode( "ccdTempState\0", &(fieldStruct.camCcdTempState) );


  /* implement hierarchy */
  camCamera->addChild( camConnectedState );
  camCamera->addChild( camCcdTempState );

  camOptics->addChild( camOptConnectedState );
  camOptics->addChild( camShutterState );
  camOptics->addChild( camFilterState );

  camObs->addChild( camInProgress );
  camObs->addChild( camLastPicName );
  camObs->addChild( camLastPicPath );

  camHardware->addChild( camOptics );
  camHardware->addChild( camCamera );

  this->addChild( camUpdated );
  this->addChild( camObs );
  this->addChild( camHardware );

  return 0;
}
```

Appendix I: Mount Control VBscripts

abort.vbs

```
' SRI Telescope Automation
' Script that aborts telescope slew
' (Telescope must be in asynchronouse operation mode)

Option Explicit

' Set variables
Dim objTele             ' Stores telescope object
Dim output              ' String that will be returned

On Error Resume Next    ' Enable error handling

' Create telescope object
Set objTele = WScript.CreateObject("TheSky6.RASCOMTele")

objTele.Abort()                ' Home the mount
If Err.number <> 0 Then           ' Check whether error was encountered
    output = "Mount Error!"       ' Return error if encountered
Else
    output = "0"
End If

Set objTele = Nothing   ' Discard telescope object to deallocate memory
WScript.Echo(output)    ' Write the output to standard out
```

connect.vbs

```
' SRI Telescope Automation
' Script that starts TheSky6 and connects to mount

Option Explicit

' Set variables
Dim objTele             ' Stores telescope object
Dim output              ' String that will be returned

On Error Resume Next    ' Enable error handling

' Create telescope object
Set objTele = WScript.CreateObject("TheSky6.RASCOMTele")

objTele.Connect()                                                ' Try to Connect to the mount
If (Err.number <> 0) Or (objTele.IsConnected = 0) Then           ' Check whether error was
encountered
    output = "Mount Error!"                                      ' Return error if encountered
Else
    Call objTele.SetTracking(0, 1, 0, 0)       ' disables tracking so mount won't move after slew
    objTele.Asynchronous = 1                   ' sets telescope slews to be asynchronous
    output = "0"                               ' Return 0
End If

Set objTele = Nothing   ' Discard telescope object to deallocate memory
WScript.Echo(output)    ' Write the output to standard out
```

getposition.vbs

```
' SRI Telescope Automation
' Script that returns the current position of the telescope

Option Explicit

' Set variables
Dim objTele             ' Stores telescope object
Dim output              ' String that will be returned

' Create variables that hold desired position
```

```
Dim aziDeg
Dim aziMin
Dim aziSec
Dim altDeg
Dim altMin
Dim altSec
Dim Azimuth
Dim Altitude

On Error Resume Next    ' Enable error handling

' Function to convert decimal position to degrees, minutes, seconds
' Adapted from Microsoft: http://support.microsoft.com/kb/213449
Sub Convert_Degree(inputDec)
        Dim Degrees
        Dim Minutes
        Dim Seconds
        'Set degree to Integer of Argument Passed
        Degrees = Int(inputDec)
        altDeg = Degrees
        'Set minutes to 60 times the number to the right
        'of the decimal for the variable Decimal_Deg
        Minutes = (inputDec - Degrees) * 60
        altMin = Int(Minutes)
        'Set seconds to 60 times the number to the right of the
        'decimal for the variable Minute
        Seconds = ((Minutes - Int(Minutes)) * 60)
        altSec = Round(Seconds)

        If altSec = 60 Then
               altSec = 0
               altMin = altMin + 1
        End If

        If altMin = 60 Then
               altMin = 0
               altDeg = altDeg + 1
        End If

        If altDeg = 360 Then
               altDeg = 0
        End If

End Sub

' Create telescope object
Set objTele = WScript.CreateObject("TheSky6.RASCOMTele")

' Try to get current position
objTele.getAzAlt()
If Err.number <> 0 Then           ' Check whether error was encountered
    output = "Mount Error!"       ' Return error if encountered
Else
    Convert_Degree(objTele.dAz)    ' Get Azimuth and convert to Deg Min Sec
    aziDeg = altDeg                ' ConvertDegree writes to altitude variables!
    aziMin = altMin
    aziSec = altSec
    Convert_Degree(objTele.dAlt)   ' Get Altitude and convert to Deg Min Sec

    ' Send Azimuth and Altitude separated by CarriageReturn NewLine if no error
    output = "aziDeg=" & aziDeg & ";" & "aziMin=" & aziMin & ";" & "aziSec=" & aziSec & ";" & _
             "altDeg=" & altDeg & ";" & "altMin=" & altMin & ";" & "altSec=" & altSec & ";" &
Chr(0) & Chr(10)
End If

Set objTele = Nothing   ' Discard telescope object to deallocate memory
WScript.Echo(output)    ' Write the output to standard out
```

home.vbs

```
' SRI Telescope Automation
```

```
' Script that moves the telescope to home position

Option Explicit

' Set variables
Dim objTele             ' Stores telescope object
Dim output              ' String that will be returned

On Error Resume Next    ' Enable error handling

' Create telescope object
Set objTele = WScript.CreateObject("TheSky6.RASCOMTele")

objTele.FindHome()               ' Home the mount
If Err.number <> 0 Then          ' Check whether error was encountered
    output = "Mount Error!"      ' Return error if encountered
Else
    output = "0"
End If

Set objTele = Nothing   ' Discard telescope object to deallocate memory
WScript.Echo(output)    ' Write the output to standard out
```

isconnected.vbs

```
' SRI Telescope Automation
' Script that determines if the mount is connected
' Returns 1 if connected; 0 if not connected

Option Explicit

' Set variables
Dim objTele             ' Stores telescope object
Dim output              ' String that will be returned

On Error Resume Next    ' Enable error handling

' Create telescope object
Set objTele = WScript.CreateObject("TheSky6.RASCOMTele")
output = objTele.IsConnected  ' Detect whether it is connected


Set objTele = Nothing   ' Discard telescope object to deallocate memory
WScript.Echo(output)    ' Write the output to standard out
```

park.vbs

```
' SRI Telescope Automation
' Script that moves the mount to Park position and disconnects

Option Explicit

' Set variables
Dim objTele             ' Stores telescope object
Dim output              ' String that will be returned

On Error Resume Next    ' Enable error handling

' Create telescope object
Set objTele = WScript.CreateObject("TheSky6.RASCOMTele")

objTele.Park()                ' Park the mount
If Err.number <> 0 Then           ' Check whether error was encountered
    output = "Mount Error!"       ' Return error if encountered
Else
    output = "0"
End If

Set objTele = Nothing   ' Discard telescope object to deallocate memory
WScript.Echo(output)    ' Write the output to standard out
```

slewto.vbs

```
' SRI Telescope Automation
' Script that slews the mount to given Azimuth and Altitude
' Usage: cscript //nologo slewto.vbs azDeg azMin azSec altDeg altMin altSec
' Azimuth is 0-359 deg, 0-59min, 0.0-59.9 sec
' Altitude is 0-90 deg, 0-59min, 0.0-59.9 sec

Option Explicit

' Create variables
Dim objTele             ' Stores telescope object
Dim output              ' String that will be returned
Dim objArgs             ' Stores command-line arguments

' Create variables that hold desired position
Dim aziDeg
Dim aziMin
Dim aziSec
Dim altDeg
Dim altMin
Dim altSec
Dim Azimuth
Dim Altitude

On Error Resume Next    ' Enable error handling

' Function for converting string in Deg Min Sec into decimal
' Adapted from Microsoft: http://support.microsoft.com/kb/213449
Function Convert_Decimal(inputDeg)
   ' Declare the variables to be double precision floating-point.
   Dim degrees
   Dim minutes
   Dim seconds
   ' Set degree to value before "d" of Argument Passed.
   degrees = Left(inputDeg, InStr(1, inputDeg, "d") - 1)
   ' Set minutes to the value between the "d" and the "m"
   ' of the text string for the variable inputDeg divided by
   ' 60. The Val function converts the text string to a number.
   minutes = Mid(inputDeg, InStr(1, inputDeg, "d") + 2, _
             InStr(1,inputDeg, "m") - InStr(1, inputDeg, _
             "d") - 2) / 60
    ' Set seconds to the number to the right of "m" that is
    ' converted to a value and then divided by 3600.
    seconds = Mid(inputDeg, InStr(1, inputDeg, "m") + _
            2, Len(inputDeg) - InStr(1, inputDeg, "m") - 2) _
            / 3600
   Convert_Decimal = degrees + minutes + seconds
End Function


' Create telescope object
Set objTele = WScript.CreateObject("TheSky6.RASCOMTele")
set objArgs = WScript.Arguments   ' grab the command-line arguments

aziDeg = objArgs(0)
aziMin = objArgs(1)
aziSec = objArgs(2)
altDeg = objArgs(3)
altMin = objArgs(4)
altSec = objArgs(5)

' Check whether command line arguments are in correct syntax:
If Not WScript.Arguments.Count = 6 _
    Or (aziDeg Or aziMin Or aziSec Or altDeg Or altMin Or altSec) < 0 _
    Or aziDeg > 359 Or altDeg > 90 _
    Or (aziMin Or aziSec Or altMin Or altSec) > 59.999  Then  ' Check for 6 arguments
    output = output & "Syntax Error!"
    WScript.Echo(output)    ' Write the output to standard out
    Set objTele = Nothing
```

```
    WScript.Quit            ' Exit the script
End If

' Convert the input from Degrees, Minutes, Seconds, to Decimal Degrees
Azimuth = Convert_Decimal(aziDeg & "d " & aziMin & "m " & aziSec & "s")
Altitude = Convert_Decimal(altDeg & "d " & altMin & "m " & altSec & "s")

' Move the mount
Call objTele.SlewToAzAlt(Azimuth, Altitude, " ")
If Err.number <> 0 Then           ' Check whether error was encountered
    output = "Mount Error!"      ' Return error if encountered
Else
    ' Send Azimuth and Altitude separated by CarriageReturn NewLine if no error
    output = "0"
End If

Set objTele = Nothing   ' Discard telescope object to deallocate memory
WScript.Echo(output)    ' Write the output to standard out
```

Appendix J: WinView Header Format

WINHEAD.TXT

$Date: 2008/02/27 22:44:34 $

Header Structure For WinView/WinSpec (WINX) Files

The current data file used for WINX files consists of a 4100 (1004 Hex) byte header followed by the data.

Beginning with Version 2.5, many more items were added to the header to make it a complete as possible record of the data collection. This includes spectrograph and pulser information. Much of these additions were accomplished
by recycling old information which had not been used in many versions. All data files created under previous 2.x versions of WinView/WinSpec CAN still be read correctly. HOWEVER, files created under the new versions (2.5 and higher) CANNOT be read by previous versions of WinView/WinSpec OR by the CSMA software package.

****

		Decimal Byte Offset	
short	ControllerVersion	0	Hardware Version
short	LogicOutput	2	Definition of Output BNC
WORD	AmpHiCapLowNoise	4	Amp Switching Mode
WORD	xDimDet	6	Detector x dimension of chip.
short	mode	8	timing mode
float	exp_sec	10	alternitive exposure, in sec.
short	VChipXdim	14	Virtual Chip X dim
short	VChipYdim	16	Virtual Chip Y dim
WORD	yDimDet	18	y dimension of CCD or detector.
char	date[DATEMAX]	20	date
short	VirtualChipFlag	30	On/Off
char	Spare_1[2]	32	
short	noscan	34	Old number of scans - should always be -1
float	DetTemperature	36	Detector Temperature Set
short	DetType	40	CCD/DiodeArray type
WORD	xdim	42	actual # of pixels on x axis
short	stdiode	44	trigger diode
float	DelayTime	46	Used with Async Mode
WORD	ShutterControl	50	Normal, Disabled Open, Disabled Closed
short	AbsorbLive	52	On/Off
WORD	AbsorbMode	54	Reference Strip or File
short	CanDoVirtualChipFlag	56	T/F Cont/Chip able to do Virtual Chip
short	ThresholdMinLive	58	On/Off
float	ThresholdMinVal	60	Threshold Minimum Value
short	ThresholdMaxLive	64	On/Off
float	ThresholdMaxVal	66	Threshold Maximum Value
short	SpecAutoSpectroMode	70	T/F Spectrograph Used

```
  float   SpecCenterWlNm               72  Center Wavelength in Nm
  short   SpecGlueFlag                 76  T/F File is Glued
  float   SpecGlueStartWlNm            78  Starting Wavelength in Nm
  float   SpecGlueEndWlNm              82  Starting Wavelength in Nm
  float   SpecGlueMinOvrlpNm           86  Minimum Overlap in Nm
  float   SpecGlueFinalResNm           90  Final Resolution in Nm
  short   PulserType                   94  0=None, PG200=1, PTG=2, DG535=3
  short   CustomChipFlag               96  T/F Custom Chip Used
  short   XPrePixels                   98  Pre Pixels in X direction
  short   XPostPixels                 100  Post Pixels in X direction
  short   YPrePixels                  102  Pre Pixels in Y direction
  short   YPostPixels                 104  Post Pixels in Y direction
  short   asynen                      106  asynchron enable flag  0 = off
  short   datatype                    108  experiment datatype
                                            0 =   float (4 bytes)
                                            1 =   long (4 bytes)
                                            2 =   short (2 bytes)
                                            3 =   unsigned short (2 bytes)
  short   PulserMode                  110  Repetitive/Sequential
  WORD    PulserOnChipAccums          112  Num PTG On-Chip Accums
  DWORD   PulserRepeatExp             114  Num Exp Repeats (Pulser SW Accum)
  float   PulseRepWidth               118  Width Value for Repetitive pulse
(usec)
  float   PulseRepDelay               122  Width Value for Repetitive pulse
(usec)
  float   PulseSeqStartWidth          126  Start Width for Sequential pulse
(usec)
  float   PulseSeqEndWidth            130  End Width for Sequential pulse
(usec)
  float   PulseSeqStartDelay          134  Start Delay for Sequential pulse
(usec)
  float   PulseSeqEndDelay            138  End Delay for Sequential pulse
(usec)
  short   PulseSeqIncMode             142  Increments: 1=Fixed,
2=Exponential
  short   PImaxUsed                   144  PI-Max type controller flag
  short   PImaxMode                   146  PI-Max mode
  short   PImaxGain                   148  PI-Max Gain
  short   BackGrndApplied             150  1 if background subtraction done
  short   PImax2nsBrdUsed             152  T/F PI-Max 2ns Board Used
  WORD    minblk                      154  min. # of strips per skips
  WORD    numminblk                   156  # of min-blocks before geo skps
  short   SpecMirrorLocation[2]       158  Spectro Mirror Location, 0=Not
Present
  short   SpecSlitLocation[4]         162  Spectro Slit Location, 0=Not
Present
  short   CustomTimingFlag            170  T/F Custom Timing Used
  char    ExperimentTimeLocal[TIMEMAX] 172  Experiment Local Time as hhmmss\0
  char    ExperimentTimeUTC[TIMEMAX]  179  Experiment UTC Time as hhmmss\0
  short   ExposUnits                  186  User Units for Exposure
  WORD    ADCoffset                   188  ADC offset
  WORD    ADCrate                     190  ADC rate
  WORD    ADCtype                     192  ADC type
  WORD    ADCresolution               194  ADC resolution
  WORD    ADCbitAdjust                196  ADC bit adjust
  WORD    gain                        198  gain
  char    Comments[5][COMMENTMAX]     200  File Comments
```

```
  WORD    geometric                   600  geometric ops: rotate 0x01,
                                            reverse 0x02, flip 0x04
  char    xlabel[LABELMAX]            602  intensity display string
  WORD    cleans                      618  cleans
  WORD    NumSkpPerCln                620  number of skips per clean.
  short   SpecMirrorPos[2]            622  Spectrograph Mirror Positions
  float   SpecSlitPos[4]              626  Spectrograph Slit Positions
  short   AutoCleansActive            642  T/F
  short   UseContCleansInst           644  T/F
  short   AbsorbStripNum              646  Absorbance Strip Number
  short   SpecSlitPosUnits            648  Spectrograph Slit Position Units
  float   SpecGrooves                 650  Spectrograph Grating Grooves
  short   srccmp                      654  number of source comp. diodes
  WORD    ydim                        656  y dimension of raw data.
  short   scramble                    658  0=scrambled,1=unscrambled
  short   ContinuousCleansFlag        660  T/F Continuous Cleans Timing
Option
  short   ExternalTriggerFlag         662  T/F External Trigger Timing
Option
  long    lnoscan                     664  Number of scans (Early WinX)
  long    lavgexp                     668  Number of Accumulations
  float   ReadoutTime                 672  Experiment readout time
  short   TriggeredModeFlag           676  T/F Triggered Timing Option
  char    Spare_2[10]                 678
  char    sw_version[FILEVERMAX]      688  Version of SW creating this file
  short   type                        704   1 = new120 (Type II)
                                            2 = old120 (Type I )
                                            3 = ST130
                                            4 = ST121
                                            5 = ST138
                                            6 = DC131 (PentaMax)
                                            7 = ST133 (MicroMax/SpectroMax)
                                            8 = ST135 (GPIB)
                                            9 = VICCD
                                           10 = ST116 (GPIB)
                                           11 = OMA3 (GPIB)
                                           12 = OMA4
  short   flatFieldApplied            706  1 if flat field was applied.
  char    Spare_3[16]                 708
  short   kin_trig_mode               724  Kinetics Trigger Mode
  char    dlabel[LABELMAX]            726  Data label.
  char    Spare_4[436]                742
  char    PulseFileName[HDRNAMEMAX]  1178  Name of Pulser File with
                                            Pulse Widths/Delays (for Z-
Slice)
  char    AbsorbFileName[HDRNAMEMAX] 1298 Name of Absorbance File (if File
Mode)
  DWORD   NumExpRepeats              1418  Number of Times experiment
repeated
  DWORD   NumExpAccums               1422  Number of Time experiment
accumulated
  short   YT_Flag                    1426  Set to 1 if this file contains YT
data
  float   clkspd_us                  1428  Vert Clock Speed in micro-sec
  short   HWaccumFlag                1432  set to 1 if accum done by
Hardware.
  short   StoreSync                  1434  set to 1 if store sync used
```

```
  short   BlemishApplied             1436  set to 1 if blemish removal
applied
  short   CosmicApplied              1438  set to 1 if cosmic ray removal
applied
  short   CosmicType                 1440  if cosmic ray applied, this is
type
  float   CosmicThreshold            1442  Threshold of cosmic ray removal.
  long    NumFrames                  1446  number of frames in file.
  float   MaxIntensity               1450  max intensity of data (future)
  float   MinIntensity               1454  min intensity of data (future)
  char    ylabel[LABELMAX]           1458  y axis label.
  WORD    ShutterType                1474  shutter type.
  float   shutterComp                1476  shutter compensation time.
  WORD    readoutMode                1480  readout mode, full,kinetics, etc
  WORD    WindowSize                 1482  window size for kinetics only.
  WORD    clkspd                     1484  clock speed for kinetics & frame
transfer
  WORD    interface_type             1486  computer interface
                                            (isa-taxi, pci, eisa, etc.)
  short   NumROIsInExperiment        1488  May be more than the 10 allowed
in
                                            this header (if 0, assume 1)
  char    Spare_5[16]                1490
  WORD    controllerNum              1506  if multiple controller system
will
                                            have controller number data came
from.
                                            this is a future item.
  WORD    SWmade                     1508  Which software package created
this file
  short   NumROI                     1510  number of ROIs used. if 0 assume
1.


--------------------------------------------------------------------------
--

                        ROI entries   (1512 - 1631)

  struct ROIinfo
  {
    WORD  startx                            left x start value.
    WORD  endx                              right x value.
    WORD  groupx                            amount x is binned/grouped in hw.
    WORD  starty                            top y start value.
    WORD  endy                              bottom y value.
    WORD  groupy                            amount y is binned/grouped in hw.
  } ROIinfoblk[ROIMAX]
                                            ROI Starting Offsets:

                                              ROI  1  =  1512
                                              ROI  2  =  1524
                                              ROI  3  =  1536
                                              ROI  4  =  1548
                                              ROI  5  =  1560
                                              ROI  6  =  1572
                                              ROI  7  =  1584
```

```
                                          ROI  8  =  1596
                                          ROI  9  =  1608
                                          ROI 10  =  1620

------------------------------------------------------------------------------
--

  char    FlatField[HDRNAMEMAX]       1632  Flat field file name.
  char    background[HDRNAMEMAX]      1752  background sub. file name.
  char    blemish[HDRNAMEMAX]         1872  blemish file name.
  float   file_header_ver             1992  version of this file header
  char    YT_Info[1000]               1996-2995  Reserved for YT information
  long    WinView_id                  2996  == 0x01234567L if file created by
WinX

------------------------------------------------------------------------------
--

                        START OF X CALIBRATION STRUCTURE (3000 - 3488)

  double  offset                      3000  offset for absolute data scaling
  double  factor                      3008  factor for absolute data scaling
  char    current_unit                3016  selected scaling unit
  char    reserved1                   3017  reserved
  char    string[40]                  3018  special string for scaling
  char    reserved2[40]               3058  reserved
  char    calib_valid                 3098  flag if calibration is valid
  char    input_unit                  3099  current input units for
                                            "calib_value"
  char    polynom_unit                3100  linear UNIT and used
                                            in the "polynom_coeff"
  char    polynom_order               3101  ORDER of calibration POLYNOM
  char    calib_count                 3102  valid calibration data pairs
  double  pixel_position[10]          3103  pixel pos. of calibration data
  double  calib_value[10]             3183  calibration VALUE at above pos
  double  polynom_coeff[6]            3263  polynom COEFFICIENTS
  double  laser_position              3311  laser wavenumber for relativ WN
  char    reserved3                   3319  reserved
  BYTE    new_calib_flag              3320  If set to 200, valid label below
  char    calib_label[81]             3321  Calibration label (NULL term'd)
  char    expansion[87]               3402  Calibration Expansion area

------------------------------------------------------------------------------
--

                        START OF Y CALIBRATION STRUCTURE (3489 - 3977)

  double  offset                      3489  offset for absolute data scaling
  double  factor                      3497  factor for absolute data scaling
  char    current_unit                3505  selected scaling unit
  char    reserved1                   3506  reserved
  char    string[40]                  3507  special string for scaling
  char    reserved2[40]               3547  reserved
  char    calib_valid                 3587  flag if calibration is valid
  char    input_unit                  3588  current input units for
                                            "calib_value"
  char    polynom_unit                3589  linear UNIT and used
```

```
                                                in the "polynom_coeff"
  char    polynom_order               3590  ORDER of calibration POLYNOM
  char    calib_count                 3591  valid calibration data pairs
  double  pixel_position[10]          3592  pixel pos. of calibration data
  double  calib_value[10]             3672  calibration VALUE at above pos
  double  polynom_coeff[6]            3752  polynom COEFFICIENTS
  double  laser_position              3800  laser wavenumber for relativ WN
  char    reserved3                   3808  reserved
  BYTE    new_calib_flag              3809  If set to 200, valid label below
  char    calib_label[81]             3810  Calibration label (NULL term'd)
  char    expansion[87]               3891  Calibration Expansion area

                         END OF CALIBRATION STRUCTURES

--------------------------------------------------------------------------
--

  char    Istring[40]                 3978  special intensity scaling string
  char    Spare_6[25]                 4018
  BYTE    SpecType                    4043  spectrometer type (acton, spex,
etc.)
  BYTE    SpecModel                   4044  spectrometer model (type
dependent)
  BYTE    PulseBurstUsed              4045  pulser burst mode on/off
  DWORD   PulseBurstCount             4046  pulser triggers per burst
  double  ulseBurstPeriod             4050  pulser burst period (in usec)
  BYTE    PulseBracketUsed            4058  pulser bracket pulsing on/off
  BYTE    PulseBracketType            4059  pulser bracket pulsing type
  double  PulseTimeConstFast          4060  pulser slow exponential time
constant (in usec)
  double  PulseAmplitudeFast          4068  pulser fast exponential amplitude
constant
  double  PulseTimeConstSlow          4076  pulser slow exponential time
constant (in usec)
  double  PulseAmplitudeSlow          4084  pulser slow exponential amplitude
constant
  short   AnalogGain;                 4092  analog gain
  short   AvGainUsed                  4094  avalanche gain was used
  short   AvGain                      4096  avalanche gain value
  short   lastvalue                   4098  Always the LAST value in the
header

                         END OF HEADER
--------------------------------------------------------------------------
--

                                      4100  Start of Data


  **************************** E.O.F.*****************************

  Definitions of array sizes:
  ---------------------------

    HDRNAMEMAX  = 120     Max char str length for file name
```

```
    USERINFOMAX = 1000    User information space
    COMMENTMAX  = 80      User comment string max length (5 comments)
    LABELMAX    = 16      Label string max length
    FILEVERMAX  = 16      File version string max length
    DATEMAX     = 10      String length of file creation date string as
ddmmmyyyy\0
    ROIMAX      = 10      Max size of roi array of structures
    TIMEMAX     = 7       Max time store as hhmmss\0

Custom Data Types used in the structure:
----------------------------------------
    BYTE = unsigned char
    WORD = unsigned short
    DWORD = unsigned long


READING DATA:
-------------
The data follows the header beginning at offset 4100.

Data is stored as sequential points.

The X, Y and Frame dimensions are determined by the header.

The X dimension of the stored data is in "xdim" ( Offset 42  ).
The Y dimension of the stored data is in "ydim" ( Offset 656 ).
The number of frames of data stored is in "NumFrames" ( Offset 1446 ).

The size of a frame (in bytes) is:

One frame size = xdim x ydim x (datatype Offset 108)
```

Appendix K: MATLAB Code to Generate Generic WinView Header

```
cd('C:\Documents and Settings\Elizabeth Gerken\Desktop');

fid = fopen('WIDE0554_630nmFAST_30sec.SPE');
header = fread(fid, 4100);
header = header(:)';
fclose(fid);

fid = fopen('test.raw');
image = fread(fid);
image = image(:)';
fclose(fid);

composite = [header image];

composite(43)   = 0;
composite(44)   = 2; % x dimension
composite(657)  = 0;
composite(658)  = 2; % y dimension
composite(109)  = 3;
composite(110)  = 0; % experiment format
composite(1447) = 1;
composite(1448) = 0;
composite(1449) = 0;
composite(1450) = 0; % number of frames

fid = fopen('Test.SPE', 'w');
count = fwrite(fid, composite);
fclose(fid);
```

Appendix L: LodePNG Library Header File

```
/*
LodePNG version 20080202

Copyright (c) 2005-2008 Lode Vandevenne

This software is provided 'as-is', without any express or implied
warranty. In no event will the authors be held liable for any damages
arising from the use of this software.

Permission is granted to anyone to use this software for any purpose,
including commercial applications, and to alter it and redistribute it
freely, subject to the following restrictions:

    1. The origin of this software must not be misrepresented; you must not
    claim that you wrote the original software. If you use this software
    in a product, an acknowledgment in the product documentation would be
    appreciated but is not required.

    2. Altered source versions must be plainly marked as such, and must not be
    misrepresented as being the original software.

    3. This notice may not be removed or altered from any source
    distribution.
*/

#ifndef LODEPNG_H
#define LODEPNG_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* ////////////////////////////////////////////////////////////////////////// */
/* Code Sections                                                              */
/* ////////////////////////////////////////////////////////////////////////// */

/*
The following defines can be commented to turn off certain parts of the code to
have faster compile and smaller executable size if certain parts aren't needed.
They also divide the code into different sections to more easily see the parts.
*/

#define LODEPNG_COMPILE_ZLIB            /*deflate&zlib encoder and deflate&zlib decoder*/
#define LODEPNG_COMPILE_PNG             /*png encoder and png decoder*/
#define LODEPNG_COMPILE_DECODER         /*deflate&zlib decoder and png decoder*/
#define LODEPNG_COMPILE_ENCODER         /*deflate&zlib encoder and png encoder*/
#define LODEPNG_COMPILE_DISK            /*the optional built in harddisk file loading and saving
functions*/
#define LODEPNG_COMPILE_ANCILLARY_CHUNKS /*any code or struct datamember related to chunks other
than IHDR, IDAT, PLTE, tRNS, IEND*/
#define LODEPNG_COMPILE_UNKNOWN_CHUNKS   /*handling of unknown chunks*/
/*There is also a C++ part in the code that is automatically handled by the standard __cplusplus
#define, nothing's needed for that here*/

/* ////////////////////////////////////////////////////////////////////////// */
/* LodeFlate & LodeZlib Setting structs                                       */
/* ////////////////////////////////////////////////////////////////////////// */

#ifdef LODEPNG_COMPILE_DECODER
typedef struct LodeZlib_DecompressSettings
{
  unsigned ignoreAdler32;
} LodeZlib_DecompressSettings;

extern const LodeZlib_DecompressSettings LodeZlib_defaultDecompressSettings;
void LodeZlib_DecompressSettings_init(LodeZlib_DecompressSettings* settings);
#endif /*LODEPNG_COMPILE_DECODER*/
```

```
#ifdef LODEPNG_COMPILE_ENCODER
typedef struct LodeZlib_DeflateSettings /*deflate = compress*/
{
  /*LZ77 related settings*/
  unsigned btype; /*the block type for LZ*/
  unsigned useLZ77; /*whether or not to use LZ77*/
  unsigned windowSize; /*the maximum is 32768*/
} LodeZlib_DeflateSettings;

extern const LodeZlib_DeflateSettings LodeZlib_defaultDeflateSettings;
void LodeZlib_DeflateSettings_init(LodeZlib_DeflateSettings* settings);
#endif /*LODEPNG_COMPILE_ENCODER*/

#ifdef LODEPNG_COMPILE_ZLIB

/* ////////////////////////////////////////////////////////////////////////// */
/* LodeFlate & LodeZlib                                                       */
/* ////////////////////////////////////////////////////////////////////////// */

#ifdef LODEPNG_COMPILE_DECODER
/*This function reallocs the out buffer and appends the data.
Either, *out must be NULL and *outsize must be 0, or, *out must be a valid buffer and *outsize
its size in bytes.
After using the *out data, *out must be free'd to avoid memory leaks.*/
unsigned LodeZlib_decompress(unsigned char** out, size_t* outsize, const unsigned char* in,
size_t insize, const LodeZlib_DecompressSettings* settings);
#endif /*LODEPNG_COMPILE_DECODER*/

#ifdef LODEPNG_COMPILE_ENCODER
/*This function reallocs the out buffer and appends the data.
Either, *out must be NULL and *outsize must be 0, or, *out must be a valid buffer and *outsize
its size in bytes.
After using the *out data, *out must be free'd to avoid memory leaks.*/
unsigned LodeZlib_compress(unsigned char** out, size_t* outsize, const unsigned char* in, size_t
insize, const LodeZlib_DeflateSettings* settings);
#endif /*LODEPNG_COMPILE_ENCODER*/

#endif /*LODEPNG_COMPILE_ZLIB*/

#ifdef LODEPNG_COMPILE_PNG

/* ////////////////////////////////////////////////////////////////////////// */
/* LodePNG                                                                    */
/* ////////////////////////////////////////////////////////////////////////// */

/*LodePNG_chunk functions: all these functions take as input an unsigned char* pointer
to the start of the chunk, with data until the end of the chunk.
Use the chunk functions with care! They do not check for allocated memory boundaries*/

unsigned LodePNG_chunk_length(const unsigned char* chunk); /*get the length of the data of the
chunk. Total chunk length has 12 bytes more.*/

void LodePNG_chunk_type(char type[5], const unsigned char* chunk); /*puts the 4-byte type in null
terminated string*/
unsigned char LodePNG_chunk_type_equals(const unsigned char* chunk, const char* type); /*check if
the type is the given type*/

/*properties of PNG chunks gotten from capitalization of chunk type name, as defined by the
standard*/
unsigned char LodePNG_chunk_critical(const unsigned char* chunk); /*0: ancillary chunk, 1: it's
one of the critical chunk types*/
unsigned char LodePNG_chunk_private(const unsigned char* chunk); /*0: public, 1: private*/
unsigned char LodePNG_chunk_safetocopy(const unsigned char* chunk); /*0: the chunk is unsafe to
copy, 1: the chunk is safe to copy*/

unsigned char* LodePNG_chunk_data(unsigned char* chunk); /*get pointer to the data of the chunk*/
const unsigned char* LodePNG_chunk_data_const(const unsigned char* chunk); /*get pointer to the
data of the chunk*/

unsigned LodePNG_chunk_check_crc(const unsigned char* chunk); /*returns 0 if the crc is correct,
1 if it's incorrect*/
```

```
void LodePNG_chunk_generate_crc(unsigned char* chunk); /*generates the correct CRC from the data
and puts it in the last 4 bytes of the chunk*/

/*iterate to next chunks. Note: these functions don't do bounds checking, use with care.*/
unsigned char* LodePNG_chunk_next(unsigned char* chunk);
const unsigned char* LodePNG_chunk_next_const(const unsigned char* chunk);

/*add chunks to out buffer. It reallocs the buffer to append the data.*/
unsigned char* LodePNG_append_chunk(unsigned char** out, size_t* outlength, const unsigned char*
chunk); /*appends chunk that was already created, to the data. Returns pointer to start of
appended chunk, or NULL if error happened*/
unsigned char* LodePNG_create_chunk(unsigned char** out, size_t* outlength, unsigned length,
const char* type, const unsigned char* data); /*appends new chunk to out. Returns pointer to
start of appended chunk, or NULL if error happened; may change memory address of out buffer*/


typedef struct LodePNG_InfoColor /*info about the color type of an image*/
{
  /*header (IHDR)*/
  unsigned colorType; /*color type*/
  unsigned bitDepth;  /*bits per sample*/

  /*palette (PLTE)*/
  unsigned char* palette; /*palette in RGBARGBA... order*/
  size_t palettesize; /*palette size in number of colors (amount of bytes is 4 * palettesize)*/

  /*transparent color key (tRNS)*/
  unsigned key_defined; /*is a transparent color key given?*/
  unsigned key_r;       /*red component of color key*/
  unsigned key_g;       /*green component of color key*/
  unsigned key_b;       /*blue component of color key*/
} LodePNG_InfoColor;

void LodePNG_InfoColor_init(LodePNG_InfoColor* info);
void LodePNG_InfoColor_cleanup(LodePNG_InfoColor* info);
void LodePNG_InfoColor_copy(LodePNG_InfoColor* dest, const LodePNG_InfoColor* source);

/*it's advised to use these functions instead of alloc'ing palette manually*/
void LodePNG_InfoColor_clearPalette(LodePNG_InfoColor* info);
void LodePNG_InfoColor_addPalette(LodePNG_InfoColor* info, unsigned char r, unsigned char g,
unsigned char b, unsigned char a); /*add 1 color to the palette*/

/*additional color info*/
unsigned LodePNG_InfoColor_getBpp(const LodePNG_InfoColor* info);      /*bits per pixel*/
unsigned LodePNG_InfoColor_getChannels(const LodePNG_InfoColor* info); /*amount of channels*/
unsigned LodePNG_InfoColor_isGreyscaleType(const LodePNG_InfoColor* info); /*is it a greyscale
type? (colorType 0 or 4)*/
unsigned LodePNG_InfoColor_isAlphaType(const LodePNG_InfoColor* info);     /*has it an alpha
channel? (colorType 2 or 6)*/

#ifdef LODEPNG_COMPILE_ANCILLARY_CHUNKS

typedef struct LodePNG_Time /*LodePNG's encoder does not generate the current time. To make it
add a time chunk the correct time has to be provided*/
{
  unsigned      year;    /*2 bytes*/
  unsigned char month;   /*1-12*/
  unsigned char day;     /*1-31*/
  unsigned char hour;    /*0-23*/
  unsigned char minute;  /*0-59*/
  unsigned char second;  /*0-60 (to allow for leap seconds)*/
} LodePNG_Time;

typedef struct LodePNG_Text /*non-international text*/
{
  size_t num;
  char** keys; /*the keyword of a text chunk (e.g. "Comment")*/
  char** strings; /*the actual text*/
} LodePNG_Text;

void LodePNG_Text_init(LodePNG_Text* text);
```

```
void LodePNG_Text_cleanup(LodePNG_Text* text);
void LodePNG_Text_copy(LodePNG_Text* dest, const LodePNG_Text* source);

/*it's advised to use these functions instead of alloc'ing the char**s manually*/
void LodePNG_Text_clear(LodePNG_Text* text);
void LodePNG_Text_add(LodePNG_Text* text, const char* key, const char* str); /*push back both
texts at once*/


typedef struct LodePNG_IText /*international text*/
{
  size_t num;
  char** keys; /*the English keyword of the text chunk (e.g. "Comment")*/
  char** langtags; /*the language tag for this text's international language, ISO/IEC 646 string,
e.g. ISO 639 language tag*/
  char** transkeys; /*keyword translated to the international language - UTF-8 string*/
  char** strings; /*the actual international text - UTF-8 string*/
} LodePNG_IText;

void LodePNG_IText_init(LodePNG_IText* text);
void LodePNG_IText_cleanup(LodePNG_IText* text);
void LodePNG_IText_copy(LodePNG_IText* dest, const LodePNG_IText* source);

/*it's advised to use these functions instead of alloc'ing the char**s manually*/
void LodePNG_IText_clear(LodePNG_IText* text);
void LodePNG_IText_add(LodePNG_IText* text, const char* key, const char* langtag, const char*
transkey, const char* str); /*push back the 4 texts of 1 chunk at once*/

#endif /*LODEPNG_COMPILE_ANCILLARY_CHUNKS*/

#ifdef LODEPNG_COMPILE_UNKNOWN_CHUNKS

typedef struct LodePNG_UnknownChunks /*unknown chunks read from the PNG, or extra chunks the user
wants to have added in the encoded PNG*/
{
  /*there are 3 buffers, one for each position in the PNG where unknown chunks can appear
  each buffer contains all unknown chunks for that position consecutively
  The 3 buffers are the unknown chunks between certain critical chunks:
  0: ihdr-plte, 1: plte-idat, 2: idat-iend*/
  unsigned char* data[3];
  size_t datasize[3]; /*size in bytes of the unknown chunks, given for protection*/

} LodePNG_UnknownChunks;

void LodePNG_UnknownChunks_init(LodePNG_UnknownChunks* chunks);
void LodePNG_UnknownChunks_cleanup(LodePNG_UnknownChunks* chunks);
void LodePNG_UnknownChunks_copy(LodePNG_UnknownChunks* dest, const LodePNG_UnknownChunks* src);

#endif /*LODEPNG_COMPILE_UNKNOWN_CHUNKS*/

typedef struct LodePNG_InfoPng /*information about the PNG image, except pixels and sometimes
except width and height*/
{
  /*header (IHDR), palette (PLTE) and transparency (tRNS)*/
  unsigned width;             /*width of the image in pixels (ignored by encoder, but filled in
by decoder)*/
  unsigned height;            /*height of the image in pixels (ignored by encoder, but filled in
by decoder)*/
  unsigned compressionMethod; /*compression method of the original file*/
  unsigned filterMethod;      /*filter method of the original file*/
  unsigned interlaceMethod;   /*interlace method of the original file*/
  LodePNG_InfoColor color;    /*color type and bits, palette, transparency*/

  /*pixel data (IDAT)*/
  /*nothing stored here, the pixels are given in a separate buffer*/

#ifdef LODEPNG_COMPILE_ANCILLARY_CHUNKS

  /*suggested background color (bKGD)*/
  unsigned background_defined; /*is a suggested background color given?*/
  unsigned background_r;       /*red component of suggested background color*/
```

```
  unsigned background_g;       /*green component of suggested background color*/
  unsigned background_b;       /*blue component of suggested background color*/

  /*non-international text chunks (tEXt and zTXt)*/
  LodePNG_Text text;

  /*international text chunks (iTXt)*/
  LodePNG_IText itext;

  /*time chunk (tIME)*/
  unsigned char time_defined; /*if 0, no tIME chunk was or will be generated in the PNG image*/
  LodePNG_Time time;

  /*phys chunk (pHYs)*/
  unsigned      phys_defined; /*is pHYs chunk defined?*/
  unsigned      phys_x;
  unsigned      phys_y;
  unsigned char phys_unit; /*may be 0 (unknown unit) or 1 (metre)*/

#endif /*LODEPNG_COMPILE_ANCILLARY_CHUNKS*/

#ifdef LODEPNG_COMPILE_UNKNOWN_CHUNKS
  /*unknown chunks*/
  LodePNG_UnknownChunks unknown_chunks;
#endif /*LODEPNG_COMPILE_UNKNOWN_CHUNKS*/

} LodePNG_InfoPng;

void LodePNG_InfoPng_init(LodePNG_InfoPng* info);
void LodePNG_InfoPng_cleanup(LodePNG_InfoPng* info);
void LodePNG_InfoPng_copy(LodePNG_InfoPng* dest, const LodePNG_InfoPng* source);

typedef struct LodePNG_InfoRaw /*contains user-chosen information about the raw image data, which
is independent of the PNG image*/
{
  LodePNG_InfoColor color;
} LodePNG_InfoRaw;

void LodePNG_InfoRaw_init(LodePNG_InfoRaw* info);
void LodePNG_InfoRaw_cleanup(LodePNG_InfoRaw* info);
void LodePNG_InfoRaw_copy(LodePNG_InfoRaw* dest, const LodePNG_InfoRaw* source);

/*
LodePNG_convert:
Converts from any color type to 24-bit or 32-bit (later maybe more supported). return value =
LodePNG error code
The out buffer must have (w * h * bpp + 7) / 8, where bpp is the bits per pixel of the output
color type (LodePNG_InfoColor_getBpp)
*/
unsigned LodePNG_convert(unsigned char* out, const unsigned char* in, LodePNG_InfoColor* infoOut,
LodePNG_InfoColor* infoIn, unsigned w, unsigned h);

#ifdef LODEPNG_COMPILE_DECODER

typedef struct LodePNG_DecodeSettings
{
  LodeZlib_DecompressSettings zlibsettings; /*in here is the setting to ignore Adler32
checksums*/

  unsigned ignoreCrc; /*ignore CRC checksums*/
  unsigned color_convert; /*whether to convert the PNG to the color type you want. Default: yes*/

#ifdef LODEPNG_COMPILE_ANCILLARY_CHUNKS
  unsigned readTextChunks; /*if false but rememberUnknownChunks is true, they're stored in the
unknown chunks*/
#endif /*LODEPNG_COMPILE_ANCILLARY_CHUNKS*/

#ifdef LODEPNG_COMPILE_UNKNOWN_CHUNKS
  unsigned rememberUnknownChunks; /*store all bytes from unknown chunks in the InfoPng (off by
default, useful for a png editor)*/
#endif /*LODEPNG_COMPILE_UNKNOWN_CHUNKS*/
```

```
} LodePNG_DecodeSettings;

void LodePNG_DecodeSettings_init(LodePNG_DecodeSettings* settings);

typedef struct LodePNG_Decoder
{
  LodePNG_DecodeSettings settings;
  LodePNG_InfoRaw infoRaw;
  LodePNG_InfoPng infoPng; /*info of the PNG image obtained after decoding*/
  unsigned error;
} LodePNG_Decoder;

void LodePNG_Decoder_init(LodePNG_Decoder* decoder);
void LodePNG_Decoder_cleanup(LodePNG_Decoder* decoder);
void LodePNG_Decoder_copy(LodePNG_Decoder* dest, const LodePNG_Decoder* source);

/*decoding functions*/
/*This function mallocs the out buffer for you and stores the size in *outsize. After using the
*out data, *out must be free'd to avoid memory leaks.*/
void LodePNG_decode(LodePNG_Decoder* decoder, unsigned char** out, size_t* outsize, const
unsigned char* in, size_t insize);
unsigned LodePNG_decode32(unsigned char** out, unsigned* w, unsigned* h, const unsigned char* in,
size_t insize); /*return value is error*/
#ifdef LODEPNG_COMPILE_DISK
unsigned LodePNG_decode32f(unsigned char** out, unsigned* w, unsigned* h, const char* filename);
#endif /*LODEPNG_COMPILE_DISK*/
void LodePNG_inspect(LodePNG_Decoder* decoder, const unsigned char* in, size_t size); /*read the
png header*/

#endif /*LODEPNG_COMPILE_DECODER*/

#ifdef LODEPNG_COMPILE_ENCODER

typedef struct LodePNG_EncodeSettings
{
  LodeZlib_DeflateSettings zlibsettings; /*settings for the zlib encoder, such as window
size, ...*/

  unsigned autoLeaveOutAlphaChannel; /*automatically use color type without alpha instead of
given one, if given image is opaque*/
  unsigned force_palette; /*force creating a PLTE chunk if colortype is 2 or 6 (= a suggested
palette). If colortype is 3, PLTE is _always_ created.*/
#ifdef LODEPNG_COMPILE_ANCILLARY_CHUNKS
  unsigned add_id; /*add LodePNG version as text chunk*/
  unsigned text_compression; /*encode text chunks as zTXt chunks instead of tEXt chunks, and use
compression in iTXt chunks*/
#endif /*LODEPNG_COMPILE_ANCILLARY_CHUNKS*/
} LodePNG_EncodeSettings;

void LodePNG_EncodeSettings_init(LodePNG_EncodeSettings* settings);

typedef struct LodePNG_Encoder
{
  LodePNG_EncodeSettings settings;
  LodePNG_InfoPng infoPng; /*the info specified by the user may not be changed by the encoder.
The encoder will try to generate a PNG close to the given info.*/
  LodePNG_InfoRaw infoRaw; /*put the properties of the input raw image in here*/
  unsigned error;
} LodePNG_Encoder;

void LodePNG_Encoder_init(LodePNG_Encoder* encoder);
void LodePNG_Encoder_cleanup(LodePNG_Encoder* encoder);
void LodePNG_Encoder_copy(LodePNG_Encoder* dest, const LodePNG_Encoder* source);

/*This function mallocs the out buffer for you and stores the size in *outsize. After using the
*out data, *out must be free'd to avoid memory leaks.*/
void LodePNG_encode(LodePNG_Encoder* encoder, unsigned char** out, size_t* outsize, const
unsigned char* image, unsigned w, unsigned h);
unsigned LodePNG_encode32(unsigned char** out, size_t* outsize, const unsigned char* image,
unsigned w, unsigned h); /*return value is error*/
#ifdef LODEPNG_COMPILE_DISK
```

```
unsigned LodePNG_encode32f(const char* filename, const unsigned char* image, unsigned w, unsigned
h);
#endif /*LODEPNG_COMPILE_DISK*/

#endif /*LODEPNG_COMPILE_ENCODER*/

#endif /*LODEPNG_COMPILE_PNG*/

#ifdef LODEPNG_COMPILE_DISK
/*global functions allowing to load and save a file from/to harddisk*/
/*This function mallocs the out buffer for you and stores the size in *outsize. After using the
*out data, *out must be free'd to avoid memory leaks.*/
void LodePNG_loadFile(unsigned char** out, size_t* outsize, const char* filename);
void LodePNG_saveFile(const unsigned char* buffer, size_t buffersize, const char* filename);
#endif /*LODEPNG_COMPILE_DISK*/

#ifdef __cplusplus
/*
C++ RAII wrapper:
-introduces RAII thanks to ctors and dtors of Decoder and Encoder class
-introduces std::vector versions of the encode and decode functions
-brings back the interface almost completely identical to the original C++ version of LodePNG,
 except for the std::vector version of palette and std::strings of text chunks
*/
#include <vector>
#include <string>
#include <fstream>

#ifdef LODEPNG_COMPILE_ZLIB

namespace LodeZlib
{
#ifdef LODEPNG_COMPILE_DECODER
  unsigned decompress(std::vector<unsigned char>& out, const std::vector<unsigned char>& in,
const LodeZlib_DecompressSettings& settings = LodeZlib_defaultDecompressSettings);
#endif /*#ifdef LODEPNG_COMPILE_DECODER*/
#ifdef LODEPNG_COMPILE_ENCODER
  unsigned compress(std::vector<unsigned char>& out, const std::vector<unsigned char>& in, const
LodeZlib_DeflateSettings& settings = LodeZlib_defaultDeflateSettings);
#endif /*#ifdef LODEPNG_COMPILE_ENCODER*/
}

#endif /*LODEPNG_COMPILE_ZLIB*/

#ifdef LODEPNG_COMPILE_PNG

namespace LodePNG
{

#ifdef LODEPNG_COMPILE_DECODER

  class Decoder : public LodePNG_Decoder
  {
    public:

    Decoder();
    ~Decoder();
    void operator=(const LodePNG_Decoder& other);

    /*decoding functions*/
    void decode(std::vector<unsigned char>& out, const unsigned char* in, size_t insize);
    void decode(std::vector<unsigned char>& out, const std::vector<unsigned char>& in);

    void inspect(const unsigned char* in, size_t size);
    void inspect(const std::vector<unsigned char>& in);

    /*error checking after decoding*/
    bool hasError() const;
    unsigned getError() const;

    /*convenient access to some InfoPng parameters after decoding*/
```

```
    unsigned getWidth() const;
    unsigned getHeight() const;
    unsigned getBpp(); /*bits per pixel*/
    unsigned getChannels(); /*amount of channels*/
    unsigned isGreyscaleType(); /*is it a greyscale type? (colorType 0 or 4)*/
    unsigned isAlphaType(); /*has it an alpha channel? (colorType 2 or 6)*/

    const LodePNG_DecodeSettings& getSettings() const;
    LodePNG_DecodeSettings& getSettings();
    void setSettings(const LodePNG_DecodeSettings& info);

    const LodePNG_InfoPng& getInfoPng() const;
    LodePNG_InfoPng& getInfoPng();
    void setInfoPng(const LodePNG_InfoPng& info);
    void swapInfoPng(LodePNG_InfoPng& info); /*faster than copying with setInfoPng*/

    const LodePNG_InfoRaw& getInfoRaw() const;
    LodePNG_InfoRaw& getInfoRaw();
    void setInfoRaw(const LodePNG_InfoRaw& info);
  };

  /*simple functions for encoding/decoding the PNG in one call (RAW image always 32-bit)*/
  unsigned decode(std::vector<unsigned char>& out, unsigned& w, unsigned& h, const unsigned char*
in, unsigned size, unsigned colorType = 6, unsigned bitDepth = 8);
  unsigned decode(std::vector<unsigned char>& out, unsigned& w, unsigned& h, const
std::vector<unsigned char>& in, unsigned colorType = 6, unsigned bitDepth = 8);
#ifdef LODEPNG_COMPILE_DISK
  unsigned decode(std::vector<unsigned char>& out, unsigned& w, unsigned& h, const std::string&
filename, unsigned colorType = 6, unsigned bitDepth = 8);
#endif /*LODEPNG_COMPILE_DISK*/

#endif /*LODEPNG_COMPILE_DECODER*/

#ifdef LODEPNG_COMPILE_ENCODER

  class Encoder : public LodePNG_Encoder
  {
    public:

    Encoder();
    ~Encoder();
    void operator=(const LodePNG_Encoder& other);

    void encode(std::vector<unsigned char>& out, const unsigned char* image, unsigned w, unsigned
h);
    void encode(std::vector<unsigned char>& out, const std::vector<unsigned char>& image,
unsigned w, unsigned h);

    /*error checking after decoding*/
    bool hasError() const;
    unsigned getError() const;

    /*convenient direct access to some parameters of the InfoPng*/
    void clearPalette();
    void addPalette(unsigned char r, unsigned char g, unsigned char b, unsigned char a); /*add 1
color to the palette*/
#ifdef LODEPNG_COMPILE_ANCILLARY_CHUNKS
    void clearText();
    void addText(const std::string& key, const std::string& str); /*push back both texts at
once*/
    void clearIText();
    void addIText(const std::string& key, const std::string& langtag, const std::string& transkey,
const std::string& str);
#endif /*LODEPNG_COMPILE_ANCILLARY_CHUNKS*/

    const LodePNG_EncodeSettings& getSettings() const;
    LodePNG_EncodeSettings& getSettings();
    void setSettings(const LodePNG_EncodeSettings& info);

    const LodePNG_InfoPng& getInfoPng() const;
    LodePNG_InfoPng& getInfoPng();
```

```
    void setInfoPng(const LodePNG_InfoPng& info);
    void swapInfoPng(LodePNG_InfoPng& info); /*faster than copying with setInfoPng*/

    const LodePNG_InfoRaw& getInfoRaw() const;
    LodePNG_InfoRaw& getInfoRaw();
    void setInfoRaw(const LodePNG_InfoRaw& info);
  };

  unsigned encode(std::vector<unsigned char>& out, const unsigned char* in, unsigned w, unsigned
h, unsigned colorType = 6, unsigned bitDepth = 8);
  unsigned encode(std::vector<unsigned char>& out, const std::vector<unsigned char>& in, unsigned
w, unsigned h, unsigned colorType = 6, unsigned bitDepth = 8);
#ifdef LODEPNG_COMPILE_DISK
  unsigned encode(const std::string& filename, const unsigned char* in, unsigned w, unsigned h,
unsigned colorType = 6, unsigned bitDepth = 8);
  unsigned encode(const std::string& filename, const std::vector<unsigned char>& in, unsigned w,
unsigned h, unsigned colorType = 6, unsigned bitDepth = 8);
#endif /*LODEPNG_COMPILE_DISK*/

#endif /*#ifdef LODEPNG_COMPILE_ENCODER*/

#ifdef LODEPNG_COMPILE_DISK
  /*global functions allowing to load and save a file from/to harddisk*/
  void loadFile(std::vector<unsigned char>& buffer, const std::string& filename);
  void saveFile(const std::vector<unsigned char>& buffer, const std::string& filename);
#endif /*LODEPNG_COMPILE_DISK*/

} /*namespace LodePNG*/

#endif /*LODEPNG_COMPILE_PNG*/

#endif /*__cplusplus C++ RAII wrapper*/

/*
TODO:
[ ] test if there are no leaks or exploits if a function returns in the middle due to an error
[ ] LZ77 encoder more like the one described in zlib - to make sure it's patentfree
[ ] converting color to 16-bit types
[ ] read all public PNG chunk types (but never let the color profile and gamma ones ever touch
RGB values, that is very annoying for textures as well as images in a browser)
[X] add option to decoder to store ignored chunks in LodePNG_InfoPng, and let encoder include
those in the result
[X] encoding PNGs with Adam7 interlace
[ ] make sure encoder generates no chunks with size > (2^31)-1
[ ] partial decoding (stream processing)
[ ] let the "isFullyOpaque" function check color keys and transparent palettes too
[ ] better name for "codes", "codesD", "codelengthcodes", "clcl" and "lldl"
[X] support zTXt chunks
[X] support iTXt chunks
[ ] check compatibility with vareous compilers (done but needs to be redone for every newer
version)
[ ] don't stop decoding on errors like 69, 57, 58 (make warnings that the decoder stores in the
error at the very end? and make some errors just let it stop with this one chunk but still do the
next ones)
[ ] make option to choose if the raw image with non multiple of 8 bits per scanline should have
padding bits or not, if people like storing raw images that way
*/

#endif

/*
LodePNG Documentation
---------------------

0. table of contents
--------------------

  1. about
   1.1. supported features
   1.2. features not supported
  2. C and C++ version
```

1. about

PNG is a file format to store raster images losslessly with good compression,
supporting different color types. It can be implemented in a patent-free way.

LodePNG is a PNG codec according to the Portable Network Graphics (PNG)
Specification (Second Edition) - W3C Recommendation 10 November 2003.

The specifications used are:

*) Portable Network Graphics (PNG) Specification (Second Edition):
     http://www.w3.org/TR/2003/REC-PNG-20031110
*) RFC 1950 ZLIB Compressed Data Format version 3.3:
     http://www.gzip.org/zlib/rfc-zlib.html
*) RFC 1951 DEFLATE Compressed Data Format Specification ver 1.3:
     http://www.gzip.org/zlib/rfc-deflate.html

The most recent version of LodePNG can currently be found at
http://members.gamedev.net/lode/projects/LodePNG/

LodePNG works both in C (ISO C90) and C++, with a C++ wrapper that adds
extra functionality.

LodePNG exists out of two files:
-lodepng.h: the header file for both C and C++
-lodepng.c(pp): give it the name lodepng.c or lodepng.cpp depending on your usage
Optionally, LodePNG also has the files lodepng_examples.c and lodepng_examples.cpp

LodePNG is simple but only supports the basic requirements. To achieve
simplicity, the following design choices were made: There are no dependencies
on any external library. To decode PNGs, there's a Decoder struct or class that
can convert any PNG file data into an RGBA image buffer with a single function
call. To encode PNGs, there's an Encoder struct or class that can convert image
data into PNG file data with a single function call. To read and write files,
simple functions to convert the files to/from buffers in memory.

This all makes LodePNG suitable for loading textures in games, raytracers,
intros, ..., or for loading images into programs that require them only for
simple usage. It's less suitable for full fledged image editors, loading PNGs
over network (since this decoder requires all the image data to be available
before the decoding can begin), life-critical systems, ... Even though it
contains a conformant decoder and encoder, it's still not a conformant editor,
because unknown chunks are discarded.

1.1. supported features

The following features are supported by the decoder:

*) decoding of PNGs with any color type, bit depth and interlace mode

*) encoding of PNGs, from any raw image to 24 or 32-bit color, or from specific raw images to any PNG color type
*) Adam7 interlace and deinterlace for any color type
*) (auto) conversion of color types, from any color type, to 24-bit, 32-bit, ...
*) loading the image from harddisk or decoding it from a buffer from other sources than harddisk
*) support for alpha channels, including translucent palettes and color key
*) zlib decompression (inflate)
*) zlib compression (deflate)
*) CRC32 and ADLER32 checksums
*) handling of unknown chunks, allowing making a PNG editor that stores custom and unknown chunks.
*) the following chunks are supported (generated/interpreted) by both encoder and decoder:
    IHDR: header information
    PLTE: color palette
    IDAT: pixel data
    IEND: the final chunk
    tRNS: transparency for palettized images
    tEXt: textual information
    zTXt: compressed textual information
    iTXt: international textual information
    bKGD: suggested background color
    pHYs: physical dimensions
    tIME: modification time

1.2. features not supported

The following features are _not_ supported:

*) some features needed to make a conformant PNG-Editor might be still missing.
*) partial loading/stream processing. All data must be available and is processed in one call.
*) The following public chunks are not supported but treated as unknown chunks by LodePNG
    cHRM, gAMA, iCCP, sRGB, sBIT, hIST, sPLT

2. C and C++ version

LodePNG is written in C (ISO C90), and has a C++ wrapper around the C version.
The C++ wrapper adds RAII, the usage of std::vectors, and convenience functions.

The C version uses buffers allocated with alloc instead that you need to free()
yourself. On top of that, you need to use init and cleanup functions for each
struct whenever using a struct from the C version to avoid exploits and memory leaks.

Both the C and the C++ version are contained in this file! The C++ code depends on
the C code, the C code works on its own.

These files work without modification for both C and C++ compilers because all the
additional C++ code is in "#ifdef __cplusplus" blocks that make C-compilers ignore
it, and all the C code is made so that it compiles both with strict ISO C90 and C++.

To use the C++ version, you need to rename the source file to lodepng.cpp (instead
of lodepng.c), and compile this with a C++ compiler.

To use the C version, you need to rename the source file to lodepng.c (instead
of lodepng.cpp), and compile this with a C compiler. Optionally, you may remove
the C++ code that is in "#ifdef __cplusplus" blocks, because that code is not
used for the C version.

3. A note about security!

In the C version of LodePNG, and in the C++ version for the "Info" structs:

For all LodePNG, LodeFlate and LodeZlib structs in C:
-if a struct has a corresponding init function, always call the init function when making a new one, to avoid exploits
-if a struct has a corresponding cleanup function, call it before the struct disappears to avoid memory leaks

-if a struct has a corresponding copy function, use the copy function instead of "=". The destination must be inited already!
-to get the effect of a copy constructor, first init, then copy
-structs will init, copy and cleanup possible member structs that they contain
-if a struct has a corresponding swap function, you can swap anything with anything, even uninited structs. This can be faster than using copy.

The C++ wrapper has classes that handle all this using RAII. More specifically, the Encoder and Decoder classes have
a constructor, destructor and operator= that use the init, cleanup and copy functions on all their members and themselves.

If you discover a possible exploit, please let me know, because they have to be eliminated at all cost.

4. "Simple" Functions

For the most simple usage cases of loading and saving a PNG image, there are some simple functions that do everything in 1 call (instead of you having to declare a struct or class).

The simple versions always use 32-bit RGBA color for the raw image, but still support loading arbitrary-colortype PNG images.

The later sections of this manual are devoted to the complex versions, where you can use other color types and conversions.

4.1 C Simple Functions

The C simple functions have a "32" or "32f" in their name, and don't take a struct as parameter, unlike the non-simple ones (see more down in the documentation).

unsigned LodePNG_decode32(unsigned char** out, unsigned* w, unsigned* h, const unsigned char* in, size_t insize);

Load PNG from given buffer.
As input, give an unsigned char* buffer gotten by loading the .png file and its size.
As output, you get a dynamically allocated buffer of large enough size, and the width and height of the image.
The buffer's size is w * h * 4. The image is in RGBA format.
The return value is the error (0 if ok).
You need to do free(out) after usage to clean up the memory.

unsigned LodePNG_decode32f(unsigned char** out, unsigned* w, unsigned* h, const char* filename);

Load PNG from disk, from file with given name.
Same as decode32, except you give a filename instead of an input buffer.

unsigned LodePNG_encode32(unsigned char** out, size_t* outsize, const unsigned char* image, unsigned w, unsigned h);

Encode PNG into buffer.
As input, give a image buffer of size w * h * 4, in RGBA format.
As output, you get a dynamically allocated buffer and its size, which is a PNG file that can directly be saved in this form to the harddisk.
The return value is the error (0 if ok).
You need to do free(out) after usage to clean up the memory.

unsigned LodePNG_encode32f(const char* filename, const unsigned char* image, unsigned w, unsigned h);

Encode PNG into file on disk with given name.
If the file exists, it's overwritten without warning!
Same parameters as encode2, except the result is stored in a file instead of a dynamic buffer.

4.2 C++ Simple Functions

```
For decoding a PNG there are:

unsigned LodePNG::decode(std::vector<unsigned char>& out, unsigned& w, unsigned& h, const
unsigned char* in, unsigned size);
unsigned LodePNG::decode(std::vector<unsigned char>& out, unsigned& w, unsigned& h, const
std::vector<unsigned char>& in);
unsigned LodePNG::decode(std::vector<unsigned char>& out, unsigned& w, unsigned& h, const
std::string& filename);

These store the pixel data as 32-bit RGBA color in the out vector, and the width
and height of the image in w and h.
The 3 functions each have a different input type: The first as unsigned char
buffer, the second as std::vector buffer, and the third allows you to give the
filename in case you want to load the PNG from disk instead of from a buffer.
The return value is the error (0 if ok).

For encoding a PNG there are:

unsigned LodePNG::encode(std::vector<unsigned char>& out, const unsigned char* in, unsigned w,
unsigned h);
unsigned LodePNG::encode(std::vector<unsigned char>& out, const std::vector<unsigned char>& in,
unsigned w, unsigned h);
unsigned LodePNG::encode(const std::string& filename, const std::vector<unsigned char>& in,
unsigned w, unsigned h);
unsigned LodePNG::encode(const std::string& filename, const unsigned char* in, unsigned w,
unsigned h);

Specify the width and height of the input image with w and h.
You can choose to get the output in an std::vector or stored in a file, and
the input can come from an std::vector or an unsigned char* buffer. The input
buffer must be in RGBA format and the size must be w * h * 4 bytes.

The first two functions append to the out buffer, they don't clear it, clear it
first before encoding into a buffer that you expect to only contain this result.

On the other hand, the functions that encode to a file will completely overwrite
the original file without warning if it exists.

The return value is the error (0 if ok).

5. Decoder
----------

This is about the LodePNG_Decoder struct in the C version, and the
LodePNG::Decoder class in the C++ version. The C++ version inherits
from the C struct and adds functions in the interface.

The Decoder class can be used to convert a PNG image to a raw image.

Usage:

-in C++:
  declare a LodePNG::Decoder
  call its decode member function with the parameters described below

-in C more needs to be done due to the lack of constructors and destructors:
  declare a LodePNG_Decoder struct
  call LodePNG_Decoder_init with the struct as parameter
  call LodePNG_Decode with the parameters described below
  after usage, call LodePNG_Decoder_cleanup with the struct as parameter
  after usage, free() the out buffer with image data that was created by the decode function

The other parameters of the decode function are:
*) out: this buffer will be filled with the raw image pixels
*) in: pointer to the PNG image data or std::vector with the data
*) size: the size of the PNG image data (not needed for std::vector version)

After decoding you need to read the width and height of the image from the
decoder, see further down in this manual to see how.

There's also an optional function "inspect". It has the same parameters as decode except
```

the "out" parameter. This function will read only the header chunk of the PNG
image, and store the information from it in the LodePNG_InfoPng (see below).
This allows knowing information about the image without decoding it. Only the
header (IHDR) information is read by this, not text chunks, not the palette, ...

During the decoding it's possible that an error can happen, for example if the
PNG image was corrupted. To check if an error happened during the last decoding,
check the value error, which is a member of the decoder struct.
In the C++ version, use hasError() and getError() of the Decoder.
The error codes are explained in another section.

Now about colors and settings...

The Decoder contains 3 components:
*) LodePNG_InfoPng: it stores information about the PNG (the input) in an LodePNG_InfoPng struct,
don't modify this one yourself
*) Settings: you can specify a few other settings for the decoder to use
*) LodePNG_InfoRaw: here you can say what type of raw image (the output) you want to get

Some of the parameters described below may be inside the sub-struct "LodePNG_InfoColor color".
In the C and C++ version, when using Info structs outside of the decoder or encoder, you need to
use their
init and cleanup functions, but normally you use the ones in the decoder that are already handled
in the init and cleanup functions of the decoder itself.

=LodePNG_InfoPng=

This contains information such as the original color type of the PNG image, text
comments, suggested background color, etc... More details about the LodePNG_InfoPng struct
are in another section.

Because the dimensions of the image are important, there are shortcuts to get them in the
C++ version: use decoder.getWidth() and decoder.getHeight().
In the C version, use decoder.infoPng.width and decoder.infoPng.height.

=LodePNG_InfoRaw=

In the LodePNG_InfoRaw struct of the Decoder, you can specify which color type you want
the resulting raw image to be. If this is different from the colorType of the
PNG, then the decoder will automatically convert the result to your LodePNG_InfoRaw
settings. Currently the following options are supported to convert to:
-colorType 6, bitDepth 8: 32-bit RGBA
-colorType 2, bitDepth 8: 24-bit RGB
-other color types if it's exactly the same as that in the PNG image

Palette of LodePNG_InfoRaw isn't used by the Decoder, when converting from palette color
to palette color, the values of the pixels are left untouched so that the colors
will change if the palette is different. Color key of LodePNG_InfoRaw is not used by the
Decoder. If setting color_convert is false then LodePNG_InfoRaw is completely ignored,
but it will be modified to match the color type of the PNG so will be overwritten.

By default, 32-bit color is used for the result.

=Settings=

The Settings can be used to ignore the errors created by invalid CRC and Adler32
chunks, and to disable the decoding of tEXt chunks.

There's also a setting color_convert, true by default. If false, no conversion
is done, the resulting data will be as it was in the PNG (after decompression)
and you'll have to puzzle the colors of the pixels together yourself using the
color type information in the LodePNG_InfoPng.

6. Encoder

This is about the LodePNG_Encoder struct in the C version, and the
LodePNG::Encoder class in the C++ version.

The Encoder class can be used to convert raw image data into a PNG image.

```
The PNG part of the encoder is working good, the zlib compression part is
becoming quite fine but not as good as the official zlib yet, because it's not
as fast and doesn't provide an as high compression ratio.

Usage:

-in C++:
  declare a LodePNG::Encoder
  call its encode member function with the parameters described below

-in C more needs to be done due to the lack of constructors and destructors:
  declare a LodePNG_Encoder struct
  call LodePNG_Encoder_init with the struct as parameter
  call LodePNG_Encode with the parameters described below
  after usage, call LodePNG_Encoder_cleanup with the struct as parameter
  after usage, free() the out buffer with PNG data that was created by the encode function

The raw image given to the encoder is an unsigned char* buffer. You also have to
specify the width and height of the raw image. The result is stored in a given
buffer. These buffers can be unsigned char* pointers, std::vectors or dynamically
allocated unsigned char* buffers that you have to free() yourself, depending on
which you use.

The parameters of the encode function are:
*) out: in this buffer the PNG file data will be stored (it will be appended)
*) in: vector of or pointer to a buffer containing the raw image
*) w and h: the width and height of the raw image in pixels

Make sure that the in buffer you provide, is big enough to contain w * h pixels
of the color type specified by the LodePNG_InfoRaw.

In the C version, you need to free() the out buffer after usage to avoid memory leaks.
In the C version, you need to use the LodePNG_Encoder_init function before using the decoder,
and the LodePNG_Encoder_cleanup function after using it.
In the C++ version, you don't need to do this since RAII takes care of it.

The encoder generates some errors but not for everything, because, unlike when
decoding a PNG, when encoding one there aren't so much parameters of the input
that can be corrupted. It's the responsibility of the user to make sure that all
preconditions are satesfied, such as giving a correct window size, giving an
existing btype, making sure the given buffer is large enough to contain an image
with the given width and height and colortype, ... The encoder can generate
some errors, see the section with the explanations of errors for those.

Like the Decoder, the Encoder has 3 components:
*) LodePNG_InfoRaw: here you say what color type of the raw image (the input) has
*) Settings: you can specify a few settings for the encoder to use
*) LodePNG_InfoPng: the same LodePNG_InfoPng struct as created by the Decoder. For the encoder,
with this you specify how you want the PNG (the output) to be.

Some of the parameters described below may be inside the sub-struct "LodePNG_InfoColor color".
In the C and C++ version, when using Info structs outside of the decoder or encoder, you need to
use their
init and cleanup functions, but normally you use the ones in the encoder that are already handled
in the init and cleanup functions of the decoder itself.

=LodePNG_InfoPng=

The Decoder class stores information about the PNG image in an LodePNG_InfoPng object. With
the Encoder you can do the opposite: you give it an LodePNG_InfoPng object, and it'll try
to match the LodePNG_InfoPng you give as close as possible in the PNG it encodes. For
example in the LodePNG_InfoPng you can specify the color type you want to use, possible
tEXt chunks you want the PNG to contain, etc... For an explanation of all the
values in LodePNG_InfoPng see a further section. Not all PNG color types are supported
by the Encoder.

Note that the encoder will only TRY to match the LodePNG_InfoPng struct you give.
Some things are ignored by the encoder. The width and height of LodePNG_InfoPng are
ignored as well, because instead the width and height of the raw image you give
in the input are used. In fact the encoder currently uses only the following
```

```
settings from it:
-colorType: the ones it supports
-text chunks, that you can add to the LodePNG_InfoPng with "addText"
-the color key, if applicable for the given color type
-the palette, if you encode to a PNG with colorType 3
-the background color: it'll add a bKGD chunk to the PNG if one is given
-the interlaceMethod: None (0) or Adam7 (1)

When encoding to a PNG with colorType 3, the encoder will generate a PLTE chunk.
If the palette contains any colors for which the alpha channel is not 255 (so
there are translucent colors in the palette), it'll add a tRNS chunk.

=LodePNG_InfoRaw=

You specify the color type of the raw image that you give to the input here,
including a possible transparent color key and palette you happen to be using in
your raw image data.

By default, 32-bit color is assumed, meaning your input has to be in RGBA
format with 4 bytes (unsigned chars) per pixel.

=Settings=

The following settings are supported (some are in sub-structs):
*) autoLeaveOutAlphaChannel: when this option is enabled, when you specify a PNG
color type with alpha channel (not to be confused with the color type of the raw
image you specify!!), but the encoder detects that all pixels of the given image
are opaque, then it'll automatically use the corresponding type without alpha
channel, resulting in a smaller PNG image.
*) btype: the block type for LZ77. 0 = uncompressed, 1 = fixed huffman tree, 2 = dynamic huffman
tree (best compression)
*) useLZ77: whether or not to use LZ77 for compressed block types
*) windowSize: the window size used by the LZ77 encoder (1 - 32768)
*) force_palette: if colorType is 2 or 6, you can make the encoder write a PLTE
   chunk if force_palette is true. This can used as suggested palette to convert
   to by viewers that don't support more than 256 colors (if those still exist)
*) add_id: add text chunk "Encoder: LodePNG <version>" to the image.
*) text_compression: default 0. If 1, it'll store texts as zTXt instead of tEXt chunks.
  zTXt chunks use zlib compression on the text. This gives a smaller result on
  large texts but a larger result on small texts (such as a single program name).
  It's all tEXt or all zTXt though, there's no separate setting per text yet.


7. color conversions
--------------------

For trickier usage of LodePNG, you need to understand about PNG color types and
about how and when LodePNG uses the settings in LodePNG_InfoPng, LodePNG_InfoRaw and Settings.

=PNG color types=

A PNG image can have many color types, ranging from 1-bit color to 64-bit color,
as well as palettized color modes. After the zlib decompression and unfiltering
in the PNG image is done, the raw pixel data will have that color type and thus
a certain amount of bits per pixel. If you want the output raw image after
decoding to have another color type, a conversion is done by LodePNG.

The PNG specification mentions the following color types:

0: greyscale, bit depths 1, 2, 4, 8, 16
2: RGB, bit depths 8 and 16
3: palette, bit depths 1, 2, 4 and 8
4: greyscale with alpha, bit depths 8 and 16
6: RGBA, bit depths 8 and 16

Bit depth is the amount of bits per color channel.

=Default Behaviour of LodePNG=

By default, the Decoder will convert the data from the PNG to 32-bit RGBA color,
no matter what color type the PNG has, so that the result can be used directly
```

as a texture in OpenGL etc... without worries about what color type the original
image has.

The Encoder assumes by default that the raw input you give it is a 32-bit RGBA
buffer and will store the PNG as either 32 bit or 24 bit depending on whether
or not any translucent pixels were detected in it.

To get the default behaviour, don't change the values of LodePNG_InfoRaw and LodePNG_InfoPng of
the encoder, and don't change the values of LodePNG_InfoRaw of the decoder.

=Color Conversions=

As explained in the sections about the Encoder and Decoder, you can specify
color types and bit depths in LodePNG_InfoPng and LodePNG_InfoRaw, to change the default
behaviour
explained above. (for the Decoder you can only specify the LodePNG_InfoRaw, because the
LodePNG_InfoPng contains what the PNG file has).

To avoid some confusion:
-the Decoder converts from PNG to raw image
-the Encoder converts from raw image to PNG
-the color type and bit depth in LodePNG_InfoRaw, are those of the raw image
-the color type and bit depth in LodePNG_InfoPng, are those of the PNG
-if the color type of the LodePNG_InfoRaw and PNG image aren't the same, a conversion
between the color types is done if the color types are supported

Supported color types:
-It's possible to load PNGs from any colortype and to save PNGs of any colorType.
-Both encoder and decoder use the same converter. So both encoder and decoder
suport the same color types at the input and the output. So the decoder supports
any type of PNG image and can convert it to certain types of raw image, while the
encoder supports any type of raw data but only certain color types for the output PNG.
-The converter can convert from _any_ input color type, to 24-bit RGB or 32-bit RGBA
-The converter can convert from greyscale input color type, to 8-bit greyscale or greyscale with
alpha
-If both color types are the same, conversion from anything to anything is possible
-Color types that are invalid according to the PNG specification are not allowed
-When converting from a type with alpha channel to one without, the alpha channel information is
discarded
-When converting from a type without alpha channel to one with, the result will be opaque except
pixels that have the same color as the color key of the input if one was given
-When converting from 16-bit bitDepth to 8-bit bitDepth, the 16-bit precision information is lost,
only the most significant byte is kept
-Converting from color to greyscale is not supported on purpose: choosing what kind of color to
greyscale conversion to do is not a decision a PNG codec should make
-Converting from/to a palette type, only keeps the indices, it ignores the colors defined in the
palette

No conversion needed...:
-If the color type of the PNG image and raw image are the same, then no
conversion is done, and all color types are supported.
-In the encoder, you can make it save a PNG with any color by giving the
LodePNG_InfoRaw and LodePNG_InfoPng the same color type.
-In the decoder, you can make it store the pixel data in the same color type
as the PNG has, by setting the color_convert setting to false. Settings in
infoRaw are then ignored.

The function LodePNG_convert does this, which is available in the interface but
normally isn't needed since the encoder and decoder already call it.

=More Notes=

In the PNG file format, if a less than 8-bit per pixel color type is used and the scanlines
have a bit amount that isn't a multiple of 8, then padding bits are used so that each
scanline starts at a fresh byte.
However: The input image you give to the encoder, and the output image you get from the decoder
will NOT have these padding bits in that case, e.g. in the case of a 1-bit image with a width
of 7 pixels, the first pixel of the second scanline will the the 8th bit of the first byte,
not the first bit of a new byte.

8. info values

```
--------------

Both the encoder and decoder use a variable of type LodePNG_InfoPng and LodePNG_InfoRaw, which
both also contain a LodePNG_InfoColor. Here's a list of each of the values stored in them:

*) info from the PNG header (IHDR chunk):

width:             width of the image in pixels
height:            height of the image in pixels
colorType:         color type of the original PNG file
bitDepth:          bits per sample
compressionMethod: compression method of the original file. Always 0.
filterMethod:      filter method of the original file. Always 0.
interlaceMethod:   interlace method of the original file. 0 is no interlace, 1 is adam7 interlace.

Note: width and height are only used as information of a decoded PNG image. When encoding one,
you don't have
to specify width and height in an LodePNG_Info struct, but you give them as parameters of the
encode function.
The rest of the LodePNG_Info struct IS used by the encoder though!

*) palette:

This is a dynamically allocated unsigned char array with the colors of the palette. The value
palettesize
indicates the amount of colors in the palette. The allocated size of the buffer is 4 *
palettesize bytes,
because there are 4 values per color: R, G, B and A. Even if less color channels are used, the
palette
is always in RGBA format, in the order RGBARGBARGBA.....

When encoding a PNG, to store your colors in the palette of the LodePNG_InfoRaw, first use
LodePNG_InfoColor_clearPalette, then for each color use LodePNG_InfoColor_addPalette.
In the C++ version the Encoder class also has the above functions available directly in its
interface.

Note that the palette information from the tRNS chunk is also already included in this palette
vector.

If you encode an image with palette, don't forget that you have to set the alpha channels (A) of
the palette
too, set them to 255 for an opaque palette. If you leave them at zero, the image will be encoded
as
fully invisible. This both for the palette in the infoRaw and the infoPng if the png is to have a
palette.

*) transparent color key

key_defined: is a transparent color key given?
key_r:       red/greyscale component of color key
key_g:       green component of color key
key_b:       blue component of color key

For greyscale PNGs, r, g and b will all 3 be set to the same.

This color is 8-bit for 8-bit PNGs, 16-bit for 16-bit per channel PNGs.

*) suggested background color

background_defined: is a suggested background color given?
background_r:       red component of sugg. background color
background_g:       green component of sugg. background color
background_b:       blue component of sugg. background color

This color is 8-bit for 8-bit PNGs, 16-bit for 16-bit PNGs

For greyscale PNGs, r, g and b will all 3 be set to the same. When encoding
the encoder writes the red one away.
For palette PNGs: When decoding, the RGB value will be stored, no a palette
index. But when encoding, specify the index of the palette in background_r,
the other two are then ignored.
```

```
The decoder pretty much ignores this background color, after all if you make a
PNG translucent normally you intend it to be used against any background, on
websites, as translucent textures in games, ... But you can get the color this
way if needed.

*) text and itext

Non-international text:

-text.keys:    a char** buffer containing the keywords (see below)
-text.strings: a char** buffer containing the texts (see below)
-text.num: the amount of texts in the above char** buffers (there may be more texts in itext)
-LodePNG_InfoText_clearText: use this to clear the texts again after you filled them in
-LodePNG_InfoText_addText: this function is used to push back a keyword and text

International text: This is stored in separate arrays! The sum text.num and itext.num is the real
amount of texts.

-itext.keys: keyword in English
-itext.langtags: ISO 639 letter code for the language
-itext.transkeys: keyword in this language
-itext.strings: the text in this language, in UTF-8
-itext.num: the amount of international texts in this PNG
-LodePNG_InfoIText_clearText: use this to clear the itexts again after you filled them in
-LodePNG_InfoIText_addText: this function is used to push back all 4 parts of an itext

Don't allocate these text buffers yourself. Use the init/cleanup functions
correctly and use addText and clearText.

In the C++ version the Encoder class also has the above functions available directly in its
interface.
The char** buffers are used like the argv parameter of a main() function, and (i)text.num takes
the role
of argc.

In a text, there must be as much keys as strings because they always form pairs. In an itext,
there must always be as much keys, langtags, transkeys and strings.

They keyword of text chunks gives a short description what the actual text
represents. There are a few standard standard keywords recognised
by many programs: Title, Author, Description, Copyright, Creation Time,
Software, Disclaimer, Warning, Source, Comment. It's allowed to use other keys.

The keyword is minimum 1 character and maximum 79 characters long. It's
discouraged to use a single line length longer than 79 characters for texts.

*) additional color info

These functions are available with longer names in the C version, and directly
in the Decoder's interface in the C++ version.

getBpp():          bits per pixel of the PNG image
getChannels():     amount of color channels of the PNG image
isGreyscaleType(): it's color type 0 or 4
isAlphaType():     it's color type 2 or 6

These values are calculated out of color type and bit depth of InfoColor.

The difference between bits per pixel and bit depth is that bit depth is the
number of bits per color channel, while a pixel can have multiple channels.

*) pHYs chunk (image dimensions)

phys_defined: if 0, there is no pHYs chunk and the values are undefined, if 1 else there is one
phys_x: pixels per unit in x direction
phys_y: pixels per unit in y direction
phys_unit: the unit, 0 is no unit (x and y only give the ratio), 1 is metre

*) tIME chunk (modification time)
```

```
time_defined: if 0, there is no tIME chunk and the values are undefined, if 1 there is one
time: this struct contains year as a 2-byte number (0-65535), month, day, hour, minute,
second as 1-byte numbers that must be in the correct range

Note: to make the encoder add a time chunk, set time_defined to 1 and fill in
the correct values in all the time parameters, LodePNG will not fill the current
time in these values itself, all it does is copy them over into the chunk bytes.


9. error values
---------------

The meanings of the LodePNG error values (encoder and decoder errors mixed
through each other):

*) 0: no error, everything went ok
*) 1: the Encoder/Decoder has done nothing yet, so error checking makes no sense yet
*) 10: while huffman decoding: end of input memory reached without endcode
*) 11: while huffman decoding: error in code tree made it jump outside of tree
*) 13: problem while processing dynamic deflate block
*) 14: problem while processing dynamic deflate block
*) 15: problem while processing dynamic deflate block
*) 16: unexisting code while processing dynamic deflate block
*) 17: while inflating: end of out buffer memory reached
*) 18: while inflating: invalid distance code
*) 19: while inflating: end of out buffer memory reached
*) 20: invalid deflate block BTYPE encountered while decoding
*) 21: NLEN is not ones complement of LEN in a deflate block
*) 22: while inflating: end of out buffer memory reached.
   This can happen if the inflated deflate data is longer than the amount of bytes required to
fill up
   all the pixels of the image, given the color depth and image dimensions. Something that
doesn't
   happen in a normal, well encoded, PNG image.
*) 23: while inflating: end of in buffer memory reached
*) 24: invalid FCHECK in zlib header
*) 25: invalid compression method in zlib header
*) 26: FDICT encountered in zlib header while it's not used for PNG
*) 27: PNG file is smaller than a PNG header
*) 28: incorrect PNG signature (the first 8 bytes of the PNG file)
   Maybe it's not a PNG, or a PNG file that got corrupted so that the header indicates the
corruption.
*) 29: first chunk is not the header chunk
*) 30: chunk length too large, chunk broken off at end of file
*) 31: illegal PNG color type or bpp
*) 32: illegal PNG compression method
*) 33: illegal PNG filter method
*) 34: illegal PNG interlace method
*) 35: chunk length of a chunk is too large or the chunk too small
*) 36: illegal PNG filter type encountered
*) 37: illegal bit depth for this color type given
*) 38: the palette is too big (more than 256 colors)
*) 39: more palette alpha values given in tRNS, than there are colors in the palette
*) 40: tRNS chunk has wrong size for greyscale image
*) 41: tRNS chunk has wrong size for RGB image
*) 42: tRNS chunk appeared while it was not allowed for this color type
*) 43: bKGD chunk has wrong size for palette image
*) 44: bKGD chunk has wrong size for greyscale image
*) 45: bKGD chunk has wrong size for RGB image
*) 46: value encountered in indexed image is larger than the palette size (bitdepth == 8). Is the
palette too small?
*) 47: value encountered in indexed image is larger than the palette size (bitdepth < 8). Is the
palette too small?
*) 48: the input data is empty. Maybe a PNG file you tried to load doesn't exist or is in the
wrong path.
*) 49: jumped past memory while generating dynamic huffman tree
*) 50: jumped past memory while generating dynamic huffman tree
*) 51: jumped past memory while inflating huffman block
*) 52: jumped past memory while inflating
*) 53: size of zlib data too small
*) 55: jumped past tree while generating huffman tree, this could be when the
```

```
       tree will have more leaves than symbols after generating it out of the
       given lenghts. They call this an oversubscribed dynamic bit lengths tree in zlib.
*) 56: given output image colorType or bitDepth not supported for color conversion
*) 57: invalid CRC encountered (checking CRC can be disabled)
*) 58: invalid ADLER32 encountered (checking ADLER32 can be disabled)
*) 59: conversion to unexisting or unsupported color type or bit depth requested by encoder or
decoder
*) 60: invalid window size given in the settings of the encoder (must be 0-32768)
*) 61: invalid BTYPE given in the settings of the encoder (only 0, 1 and 2 are allowed)
*) 62: conversion from non-greyscale color to greyscale color requested by encoder or decoder.
LodePNG
       leaves the choice of RGB to greyscale conversion formula to the user.
*) 63: length of a chunk too long, max allowed for PNG is 2147483647 bytes per chunk (2^31-1)
*) 64: the length of the "end" symbol 256 in the Huffman tree is 0, resulting in the inability of
a deflated
       block to ever contain an end code. It must be at least 1.
*) 66: the length of a text chunk keyword given to the encoder is longer than the maximum 79
bytes.
*) 67: the length of a text chunk keyword given to the encoder is smaller than the minimum 1 byte.
*) 68: tried to encode a PLTE chunk with a palette that has less than 1 or more than 256 colors
*) 69: unknown chunk type with "critical" flag encountered by the decoder
*) 70: insufficient memory error
*) 71: unexisting interlace mode given to encoder (must be 0 or 1)
*) 72: while decoding, unexisting compression method encountering in zTXt or iTXt chunk (it must
be 0)
*) 73: invalid tIME chunk size
*) 74: invalid pHYs chunk size
*) 75: no null termination char found while decoding any kind of text chunk, or wrong length
*) 76: iTXt chunk too short to contain required bytes

10. file IO
-----------

For cases where you want to load the PNG image from a file, you can use your own
file loading code, or the file loading and saving functions provided with
LodePNG. These use the same unsigned char format used by the Decoder and Encoder.

The loadFile function fills the given buffer up with the file from harddisk
with the given name.

The saveFile function saves the contents of the given buffer to the file
with given name. Warning: this overwrites the contents that were previously in
the file if it already existed, without warning.

Note that you don't have to decode a PNG image from a file, you can as well
retrieve the buffer another way in your code, because the decode function takes
a buffer as parameter, not a filename.

Both C and C++ versions of the loadFile and saveFile functions are available.
For the C version of loadFile, you need to free() the buffer after use. The
C++ versions use std::vectors so they clean themselves automatically.

11. chunks and PNG editing
--------------------------

If you want to add extra chunks to a PNG you encode, or use LodePNG for a PNG
editor that should follow the rules about handling of unknown chunks, or if you
program is able to read other types of chunks than the ones handled by LodePNG,
then that's possible with the chunk functions of LodePNG.

A PNG chunk has the following layout:

4 bytes length
4 bytes type name
length bytes data
4 bytes CRC

11.1 iterating through chunks
-----------------------------

If you have a buffer containing the PNG image data, then the first chunk (the
```

IHDR chunk) starts at byte number 8 of that buffer. The first 8 bytes are the
signature of the PNG and are not part of a chunk. But if you start at byte 8
then you have a chunk, and can check the following things of it.

NOTE: none of these functions check for memory buffer boundaries. To avoid
exploits, always make sure the buffer contains all the data of the chunks.
When using LodePNG_chunk_next, make sure the returned value is within the
allocated memory.

unsigned LodePNG_chunk_length(const unsigned char* chunk):

Get the length of the chunk's data. The total chunk length is this length + 12.

void LodePNG_chunk_type(char type[5], const unsigned char* chunk):
unsigned char LodePNG_chunk_type_equals(const unsigned char* chunk, const char* type):

Get the type of the chunk or compare if it's a certain type

unsigned char LodePNG_chunk_critical(const unsigned char* chunk):
unsigned char LodePNG_chunk_private(const unsigned char* chunk):
unsigned char LodePNG_chunk_safetocopy(const unsigned char* chunk):

Check if the chunk is critical in the PNG standard (only IHDR, PLTE, IDAT and IEND are).
Check if the chunk is private (public chunks are part of the standard, private ones not).
Check if the chunk is safe to copy. If it's not, then, when modifying data in a critical
chunk, unsafe to copy chunks of the old image may NOT be saved in the new one if your
program doesn't handle that type of unknown chunk.

unsigned char* LodePNG_chunk_data(unsigned char* chunk):
const unsigned char* LodePNG_chunk_data_const(const unsigned char* chunk):

Get a pointer to the start of the data of the chunk.

unsigned LodePNG_chunk_check_crc(const unsigned char* chunk):
void LodePNG_chunk_generate_crc(unsigned char* chunk):

Check if the crc is correct or generate a correct one.

unsigned char* LodePNG_chunk_next(unsigned char* chunk):
const unsigned char* LodePNG_chunk_next_const(const unsigned char* chunk):

Iterate to the next chunk. This works if you have a buffer with consecutive chunks. Note that
these
functions do no boundary checking of the allocated data whatsoever, so make sure there is enough
data available in the buffer to be able to go to the next chunk.

unsigned char* LodePNG_append_chunk(unsigned char** out, size_t* outlength, const unsigned char*
chunk):
unsigned char* LodePNG_create_chunk(unsigned char** out, size_t* outlength, unsigned length,
const char* type, const unsigned char* data):

These functions are used to create new chunks that are appended to the data in *out that has
length *outlength. The append function appends an existing chunk to the new data. The create
function creates a new chunk with the given parameters and appends it. Type is the 4-letter
name of the chunk.

11.2 chunks in infoPng

The LodePNG_InfoPng struct contains a struct LodePNG_UnknownChunks in it. This
struct has 3 buffers (each with size) to contain 3 types of unknown chunks:
the ones that come before the PLTE chunk, the ones that come between the PLTE
and the IDAT chunks, and the ones that come after the IDAT chunks.
It's necessary to make the distionction between these 3 cases because the PNG
standard forces to keep the ordering of unknown chunks compared to the critical
chunks, but does not force any other ordering rules.

infoPng.unknown_chunks.data[0] is the chunks before PLTE
infoPng.unknown_chunks.data[1] is the chunks after PLTE, before IDAT
infoPng.unknown_chunks.data[2] is the chunks after IDAT

The chunks in these 3 buffers can be iterated through and read by using the same
way described in the previous subchapter.

When using the decoder to decode a PNG, you can make it store all unknown chunks
if you set the option settings.rememberUnknownChunks to 1. By default, this option
is off and is 0.

The encoder will always encode unknown chunks that are stored in the infoPng. If
you need it to add a particular chunk that isn't known by LodePNG, you can use
LodePNG_append_chunk or LodePNG_create_chunk to the chunk data in
infoPng.unknown_chunks.data[x].

Chunks that are known by LodePNG should not be added in that way. E.g. to make
LodePNG add a bKGD chunk, set background_defined to true and add the correct
parameters there and LodePNG will generate the chunk.

12. compiler support

No libraries other than the current standard C library are needed to compile
LodePNG. For the C++ version, only the standard C++ library is needed on top.
Add the files lodepng.c(pp) and lodepng.h to your project, include
lodepng.h where needed, and your program can read/write PNG files.

Use optimization! For both the encoder and decoder, compiling with the best
optimizations makes a large difference.

Make sure that LodePNG is compiled with the same compiler of the same version
and with the same settings as the rest of the program, or the interfaces with
std::vectors and std::strings in C++ can be incompatible resulting in bad things.

CHAR_BITS must be 8 or higher, because LodePNG uses unsigned chars for octets.

*) gcc and g++

LodePNG is developed in gcc so this compiler is natively supported. It gives no
warnings with compiler options "-Wall -Wextra -pedantic -ansi", with gcc and g++
version 4.2.2 on Linux.

*) Mingw and Bloodshed DevC++

The Mingw compiler (a port of gcc) used by Bloodshed DevC++ for Windows is fully
supported by LodePNG.

*) Visual Studio 2005 and Visual C++ 2005 Express Edition

Versions 20070604 up to 20080107 have been tested on VS2005 and work. There are no
warnings, except two warnings about 'fopen' being deprecated. 'fopen' is a function
required by the C standard, so this warning is the fault of VS2005, it's nice of
them to enforce secure code, however the multiplatform LodePNG can't follow their
non-standard extensions. LodePNG is fully ISO C90 compliant.

If you're using LodePNG in VS2005 and don't want to see the deprecated warnings,
put this on top of lodepng.h before the inclusions: #define _CRT_SECURE_NO_DEPRECATE

*) Visual Studio 6.0

The C++ version of LodePNG was not supported by Visual Studio 6.0 because Visual
Studio 6.0 doesn't follow the C++ standard and implements it incorrectly.
The current C version of LodePNG has not been tested in VS6 but may work now.

*) Comeau C/C++

Vesion 20070107 compiles without problems on the Comeau C/C++ Online Test Drive
at http://www.comeaucomputing.com/tryitout in both C90 and C++ mode.

*) Compilers on Macintosh

I'd love to support Macintosh but don't have one available to test it on.
If it doesn't work with your compiler, maybe it can be gotten to work with the
gcc compiler for Macintosh. Someone reported that it doesn't work well at all

```
for Macintosh. All information on attempts to get it to work on Mac is welcome.

*) Other Compilers

If you encounter problems on other compilers, I'm happy to help out make LodePNG
support the compiler if it supports the ISO C90 and C++ standard well enough. If
the required modification to support the compiler requires using non standard or
lesser C/C++ code or headers, I won't support it.


13. examples
------------

This decoder and encoder example show the most basic usage of LodePNG (using the
classes, not the simple functions, which would be trivial)

More complex examples can be found in:
-lodepng_examples.c: 9 different examples in C, such as showing the image with SDL, ...
-lodepng_examples.cpp: the exact same examples in C++ using the C++ wrapper of LodePNG

13.1. decoder C++ example
-------------------------

/////////////////////////////////////////////////////////////////////////
#include "lodepng.h"
#include <iostream>

int main(int argc, char *argv[])
{
  const char* filename = argc > 1 ? argv[1] : "test.png";

  //load and decode
  std::vector<unsigned char> buffer, image;
  LodePNG::loadFile(buffer, filename); //load the image file with given filename
  LodePNG::Decoder decoder;
  decoder.decode(image, buffer.size() ? &buffer[0] : 0, (unsigned)buffer.size()); //decode the
png

  //if there's an error, display it
  if(decoder.hasError()) std::cout << "error: " << decoder.getError() << std::endl;

  //the pixels are now in the vector "image", use it as texture, draw it, ...
}

//alternative version using the "simple" function
int main(int argc, char *argv[])
{
  const char* filename = argc > 1 ? argv[1] : "test.png";

  //load and decode
  std::vector<unsigned char> image;
  unsigned w, h;
  unsigned error = LodePNG::decode(image, w, h, filename);

  //if there's an error, display it
  if(error != 0) std::cout << "error: " << error << std::endl;

  //the pixels are now in the vector "image", use it as texture, draw it, ...
}
/////////////////////////////////////////////////////////////////////////


13.2 encoder C++ example
------------------------

/////////////////////////////////////////////////////////////////////////
#include "lodepng.h"
#include <iostream>

int main(int argc, char *argv[])
{
```

```
  //check if user gave a filename
  if(argc <= 1)
  {
    std::cout << "please provide a filename to save to\n";
    return 0;
  }

  //generate some image
  std::vector<unsigned char> image;
  image.resize(512 * 512 * 4);
  for(unsigned y = 0; y < 512; y++)
  for(unsigned x = 0; x < 512; x++)
  {
    image[4 * 512 * y + 4 * x + 0] = 255 * !(x & y);
    image[4 * 512 * y + 4 * x + 1] = x ^ y;
    image[4 * 512 * y + 4 * x + 2] = x | y;
    image[4 * 512 * y + 4 * x + 3] = 255;
  }

  //encode and save
  std::vector<unsigned char> buffer;
  LodePNG::Encoder encoder;
  encoder.encode(buffer, image, 512, 512);
  LodePNG::saveFile(buffer, argv[1]);

  //the same as the 4 lines of code above, but in 1 call:
  //LodePNG::encode(argv[1], image, 512, 512);
}
////////////////////////////////////////////////////////////////////////////////

13.3 Decoder C example
----------------------

This example loads the PNG in 1 function call

#include "lodepng.h"

int main(int argc, char *argv[])
{
  unsigned error;
  unsigned char* image;
  size_t w, h;

  if(argc <= 1) return 0;

  error = LodePNG_decode3(&image, &w, &h, filename);

  free(image);
}

14. LodeZlib
------------

Also available in the interface is LodeZlib. Both C and C++ versions of these
functions are available. The interface is similar to that of the "simple" PNG
encoding and decoding functions.

LodeZlib can be used to zlib compress and decompress a buffer. It cannot be
used to create gzip files however. Also, it only supports the part of zlib
that is required for PNG, it does not support compression and decompression
with dictionaries.

15. changes
-----------

The version number of LodePNG is the date of the change given in the format
yyyymmdd.

Some changes aren't backwards compatible. Those are indicated with a (!)
symbol.
```

```
*) 02 feb 2008: support for international text chunks added (iTXt)
*) 23 jan 2008: small cleanups, and #defines to divide code in sections
*) 20 jan 2008: support for unknown chunks allowing using LodePNG for an editor.
*) 18 jan 2008: support for tIME and pHYs chunks added to encoder and decoder.
*) 17 jan 2008: ability to encode and decode compressed zTXt chunks added
    Also vareous fixes, such as in the deflate and the padding bits code.
*) 13 jan 2008: Added ability to encode Adam7-interlaced images. Improved
    filtering code of encoder.
*) 07 jan 2008: (!) changed LodePNG to use ISO C90 instead of C++. A
    C++ wrapper around this provides an interface almost identical to before.
    Having LodePNG be pure ISO C90 makes it more portable. The C and C++ code
    are together in these files but it works both for C and C++ compilers.
*) 29 dec 2007: (!) changed most integer types to unsigned int + other tweaks
*) 30 aug 2007: bug fixed which makes this Borland C++ compatible
*) 09 aug 2007: some VS2005 warnings removed again
*) 21 jul 2007: deflate code placed in new namespace separate from zlib code
*) 08 jun 2007: fixed bug with 2- and 4-bit color, and small interlaced images
*) 04 jun 2007: improved support for Visual Studio 2005: crash with accessing
    invalid std::vector element [0] fixed, and level 3 and 4 warnings removed
*) 02 jun 2007: made the encoder add a tag with version by default
*) 27 may 2007: zlib and png code separated (but still in the same file),
    simple encoder/decoder functions added for more simple usage cases
*) 19 may 2007: minor fixes, some code cleaning, new error added (error 69),
    moved some examples from here to lodepng_examples.cpp
*) 12 may 2007: palette decoding bug fixed
*) 24 apr 2007: changed the license from BSD to the zlib license
*) 11 mar 2007: very simple addition: ability to encode bKGD chunks.
*) 04 mar 2007: (!) tEXt chunk related fixes, and support for encoding
    palettized PNG images. Plus little interface change with palette and texts.
*) 03 mar 2007: Made it encode dynamic Huffman shorter  with repeat codes.
    Fixed a bug where the end code of a block had length 0 in the Huffman tree.
*) 26 feb 2007: Huffman compression with dynamic trees (BTYPE 2) now implemented
    and supported by the encoder, resulting in smaller PNGs at the output.
*) 27 jan 2007: Made the Adler-32 test faster so that a timewaste is gone.
*) 24 jan 2007: gave encoder an error interface. Added color conversion from any
    greyscale type to 8-bit greyscale with or without alpha.
*) 21 jan 2007: (!) Totally changed the interface. It allows more color types
    to convert to and is more uniform. See the manual for how it works now.
*) 07 jan 2007: Some cleanup & fixes, and a few changes over the last days:
    encode/decode custom tEXt chunks, separate classes for zlib & deflate, and
    at last made the decoder give errors for incorrect Adler32 or Crc.
*) 01 jan 2007: Fixed bug with encoding PNGs with less than 8 bits per channel.
*) 29 dec 2006: Added support for encoding images without alpha channel, and
    cleaned out code as well as making certain parts faster.
*) 28 dec 2006: Added "Settings" to the encoder.
*) 26 dec 2006: The encoder now does LZ77 encoding and produces much smaller files now.
    Removed some code duplication in the decoder. Fixed little bug in an example.
*) 09 dec 2006: (!) Placed output parameters of public functions as first parameter.
    Fixed a bug of the decoder with 16-bit per color.
*) 15 okt 2006: Changed documentation structure
*) 09 okt 2006: Encoder class added. It encodes a valid PNG image from the
    given image buffer, however for now it's not compressed.
*) 08 sep 2006: (!) Changed to interface with a Decoder class
*) 30 jul 2006: (!) LodePNG_InfoPng , width and height are now retrieved in different
    way. Renamed decodePNG to decodePNGGeneric.
*) 29 jul 2006: (!) Changed the interface: image info is now returned as a
    struct of type LodePNG::LodePNG_Info, instead of a vector, which was a bit clumsy.
*) 28 jul 2006: Cleaned the code and added new error checks.
    Corrected terminology "deflate" into "inflate".
*) 23 jun 2006: Added SDL example in the documentation in the header, this
    example allows easy debugging by displaying the PNG and its transparency.
*) 22 jun 2006: (!) Changed way to obtain error value. Added
    loadFile function for convenience. Made decodePNG32 faster.
*) 21 jun 2006: (!) Changed type of info vector to unsigned.
    Changed position of palette in info vector. Fixed an important bug that
    happened on PNGs with an uncompressed block.
*) 16 jun 2006: Internally changed unsigned into unsigned where
    needed, and performed some optimizations.
*) 07 jun 2006: (!) Renamed functions to decodePNG and placed them
    in LodePNG namespace. Changed the order of the parameters. Rewrote the
    documentation in the header. Renamed files to lodepng.cpp and lodepng.h
```

```
*) 22 apr 2006: Optimized and improved some code
*) 07 sep 2005: (!) Changed to std::vector interface
*) 12 aug 2005: Initial release


16. contact information
-----------------------

Feel free to contact me with suggestions, problems, comments, ... concerning
LodePNG. If you encounter a PNG image that doesn't work properly with this
decoder, feel free to send it and I'll use it to find and fix the problem.

My email address is (puzzle the account and domain together with an @ symbol):
Domain: gmail dot com.
Account: lode dot vandevenne.


Copyright (c) 2005-2008 Lode Vandevenne
*/
```

Appendix M: Dome Automation Email Correspondence

Correspondence with Ashdome

From: Ashdome <ashdome@ameritech.net>
Date: Wed, Jan 16, 2008 at 5:42 AM
Subject: RE: Dome automation follow-up email
To: Jorge Alejandro <jorge@wpi.edu>, Peter Mack <pmack@astronomical.com>

Good morning Jorge & Peter,
I looked back in the files and found the dome you are referring to. It has had quite a ride getting to the final location.
1) 14'6" diameter Model REB, manually operated drop-out lower door section, manual shutter override, with a wide aperture, 54". All electrical motors are standard for Ash, 110 v, 60 cycle, 1 ph.
Contact name at the time, 2005, John Rasmussen 207 372 6390
Rasmusjohn@aol.com
2) Shipped it to Anchorage, AK and it was moved to Gakona, AK.
That's all we were allowed to know.
Jorge, on the upside, Peter should be able to automate this dome in his sleep, his company has done this many times.
Peter, watch out for the mosquitoes up there.
Richard Olson
Ash Mfg. Co.

-----Original Message-----
From: jorge.a.alejandro@gmail.com [mailto:jorge.a.alejandro@gmail.com] On Behalf Of Jorge Alejandro
Sent: Tuesday, January 15, 2008 12:51 PM
To: Ashdome
Subject: Re: Dome automation followup email

Hello Peter,

I contacted you last week about automating a telescope dome located in Alaska (you seemed to remember the installation). I am emailing you to request some documentation of the dome hardware, specifically concerning the motors that rotate the dome and open/close the shutter. I am considering installing a MaxDome II (system which requires limit switches to be placed on the shutter and optical sensors over the motor gears. If any of this is already installed as part of the system it would be great.

The dome is about 14 feet in diameter (14' 6" I think) and has a dual-door shutter (one slides over the dome and the other opens up from it's hinge at the bottom.). If you need any more information please let me know.

Thank You,

- Jorge Alejandro

On Jan 10, 2008 5:48 AM, Ashdome <ashdome@ameritech.net> wrote:
> Hello Jorge,
> I know the dome you are talking about. It was installed by the local people
> awhile ago. There is a company that has automated many of our domes very
> successfully.
> Astronomical Consultants & Equipment, Tucson, AZ www.astronomical.com 520
> 219 8722
> Dr. Peter Mack pmack@astronomical.com
> I have know idea of the cost but if they can not help you I am sure they can
> recommend someone.
> Thank you for this opportunity to be of service.
> Yours truly,
> Richard Olson
> Ash Mfg. Co.
>
>
> -----Original Message-----
> From: jorge.a.alejandro@gmail.com [mailto:jorge.a.alejandro@gmail.com] On
> Behalf Of Jorge Alejandro
> Sent: Wednesday, January 09, 2008 3:14 PM
> To: ashdome@ameritech.net
> Subject: Dome automation followup email
>
> Hello Mr. Olsen,
>
> I just spoke with you over the phone about automating my Ashdome. I
> currently have a 14-foot dome, located in Alaska, which is controlled
> completely from a central control panel containing various buttons and
> switches. I would like to interface this control box with a PC (any
> standard PC connection) so that I may use software to control the dome
> (preferably Software Bisque's Automadome). You mentioned you would
> provide me with the contact information of someone who may be able to
> help me with this.
>
> -Thanks
>
> Jorge Alejandro

Correspondence regarding MaxDome II installation with an Ashdome

From: David Sonnek <david.sonnek@telia.com>
Date: Sat, Jan 19, 2008 at 3:44 PM
Subject: Re: Ashdome + MaxDome II
To: Jorge Alejandro <jorge@wpi.edu>

Hi Alejandro!

Sorry for this delayed reply, I've been on a business trip. I doubt that I have much information that can help you, I have used MaxDome in a rather limited way:

- No remote capabilities - the system is manually powered on for each observing session
- I have only automated dome rotation (the shutter has to be opened manually)
- I control it through Automadome/TheSky software, have e.g. not tested the ASCOM interface.

The MaxDome hardware has been reliable so far, despite the humidity in my obs. The sensors for "home" and "rotate" are simple but adequate for the task. I just put a piece of black tape on the motor axis to count turns. It was also easy to interface the motor with the 12V output of the MaxDome. Since the motor is controlled by closing one lead for each direction, I simply put two solid-state relays "nose to nose" over the 12V output leads.

I think it's tricky to make a AshDome fully robotic - no wonder there are specialists (like Meridian Controls) that charge a lot for doing it.
One problem to solve is the power cord to the upper shutter part. It's no big deal for me, at each power-on, I "home" the dome and have set Automadome limits not to allow more than one dome turn in ordetr not to overstretch the power cord. In a remotely operated robotic mode, the system has to remember between sessions where the cord is.

Another problem in remote/robotic operation is the lower part of the shutter. From the photos, it seems that the HAART dome has "A" type shutter (the same type as I have). With this shutter, the two sections have to be physically disconnected when opening only the upper shutter part. Some kind of mechanical device has to pull the chain in the closed position and keep it stretched a moment while the upper part start going up again.
As an added complexity, my scope is mounted on a GEM, so the telescope is at different heights depending on azimuth. This makes tracking around altitude 30 deg really messy...
Luckily, most of my work is photometry on hi-alt targets, so the lower shutter is mostly closed.

The "B" type is easier to control since the lower part folds out separately - the only action the control system has to do is to open both shutters when a session begins.

If your aim is only to synchronize dome rotation with the telescope, I think MaxDome is an easy way to glue together a control PC, sensors and the AshDome az motor. For remote/robotic operation, I guess one has to include hardware that Maxdome may not be able to control.

Good luck with the project!
/David

On Tue, 2008-01-15 at 11:00 -0800, Jorge Alejandro wrote:
> Hello Sir,

>
> My name is Jorge Alejandro; I am a student at the Worcester
> Polytechnic Institute in Worcester, MA and am currently working on a
> project at SRI International in Menlo Park, CA. My project involves
> the automation of a telescope system that is part of the HAARP
> research site in Gakona, Alaska. The Ashdome in use currently has no
> PC interface, and my partners and I are considering various approaches
> for adding this interface including using a MaxDome II system. After
> doing some searching on the internet we stumbled upon the homepage for
> your observatory and noticed that you have a similar system. We were
> wondering if you could provide us with any feedback on the
> installation of the system and the overall quality. Any help at all
> would be greatly appreciated.
>
> Thanks!
>
> Jorge Alejandro

Correspondence regarding Meridian Controls

From: Bryce Bennett <bryce.bennett@rmc.ca>
Date: Mon, Jan 14, 2008 at 6:36 AM
Subject: Re: Dome Automation
To: Jorge Alejandro <jorge@wpi.edu>

Hi Jorge,

We have recently removed the Meridian Controls dome control system and
will be replacing it with a wireless system from Astronomical
Consulting. We were never very satisfied with the Meridian System -
never completely reliable - and my personal recommendation would be to
look to Astronomical Consulting or other options for your future
hardware/software for dome control.

Regards,

Bryce.

Jorge Alejandro wrote:
> Hello Bryce,
>
> My name is Jorge Alejandro, I am a student at the Worcester
> Polytechnic Institute in Worcester, Massachussets currently working on
> a project at SRI International in Menlo Park, California. My project
> involves the automation of a telescope system located in Alaska which
> is part of the HAARP research project. The dome currently in use is a
> 14-foot Ashdome with no interface to the PC (It has a simple
> control-panel with various buttons for the user). I stumbled upon the
> CASTOR homepage and saw that you are using a similar dome with
> Meridian Controls automation hardware and Automadome software. I was
> wondering if you could give me any information on this hardware and
> any feedback you have on the system.
>

> Thanks
>
> Jorge
>

--
--

Bryce Bennett, Ph.D.

Royal Military College of Canada /
Collège militaire royal du Canada

Kingston, Ontario K7K 7B4

tel: 613.541.6000 x 6080
fax: 613.541.6040