Fabra-ROA Observatory

Control System

Reference Manual

Version 0.9.0

Table of Contents

Illustration Index

1 Introduction

Welcome to the Observatory Control System for San Fernando Baker-Nunn Camera. The system allows for local and remote control of all equipment including the telescope, focuser, camera and roof. Weather conditions and power mains are monitored and shutdown procedures are performed automatically if unsafe operating conditions are imminent.

See §2.1 for more information about ObsCon, the main observatory control GUI and §2.2 for the basic camera control GUI. See §3 for a detailed list of each INDI Device and Property. See §4 for some command line client programs that use the INDI Properties. See §6 for information about how to connect the hardware.

1.1 System Overview

The control system design is a client-server architecture. Hardware and supporting services are implemented as servers. Applications such as graphical user interfaces and command line programs are clients. All communication uses TCP/IP sockets for reliable distributed operation.

The servers and clients communicate using the INDI[1] protocol. This is an XML-based protocol for passing parameters back and forth in a compact efficient format. Typical bandwidth requirements for monitoring and control of all observatory functions (except camera images) are on the order of a few tens of kbps, so even simple voice-grade modem connections are sufficient for routine remote operation.

INDI drivers are written in ANSI C for the Linux operating system. Low level hardware drivers are written for Linux kernel 2.6.13. GUIs are written in Java 1.5 for maximum portability and consistency across platforms. All GUIs have been tested on Linux under KDE, Windows XP and Mac OS 10.5. Command line programs are written in ANSI C.

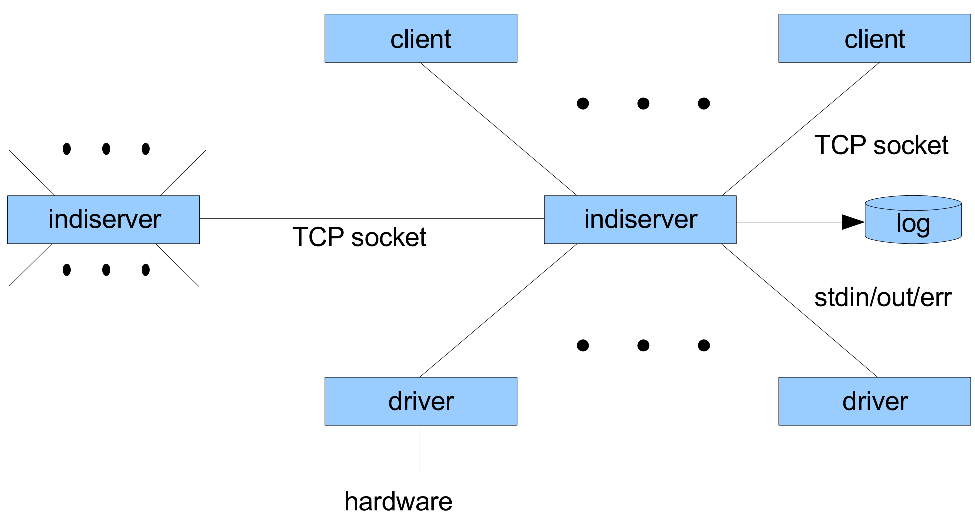

Illustration 1: INDI architecture

1 See http://www.clearskyinstitute.com/INDI/INDI.pdf

Illustration 1 shows the basic INDI data flow architecture. Each box represents one process. Each line represents either a socket connection or three UNIX pipes carrying the stdin, stdout and stderr streams.

Central to the design is the indiserver. On startup indiserver forks each driver process and arranges pipes to connect to their stdio streams. All sockets and streams carry traffic formated according to INDI XML message rules with one exception: the stderr output stream from a driver is simply copied to the log file maintained by the indiserver and can be any free-form message.

After starting each driver indiserver functions basically as a router between clients and drivers. It listens to the INDI XML messages and sends them only to interested processes based on contents of the *device* and *name* properties within the INDI message. Indiserver also serves as a process shepherd: if any driver dies, as indicated by EOF while reading from its stdin stream, indiserver will restart it and establish new stream pipes automatically.

Note that indiservers may also connect to each other. This is called chaining. Because all traffic is INDI, an indiserver may connect to another indiserver and appear to be a client in every way. When making the connection, command line arguments on the initiating indiserver specify the devices on the target indiserver with which it wishes to have visibility. The initiating indiserver will have no knowledge of other devices on the target indiserver. In this way separate indiservers may, on the one hand, share devices, or, on the other hand, intentionally hide devices from their respective connecting clients.

Within each driver is the code that implements the desired functionality for one, and only one, INDI *device*. Some drivers only provide services, such as target prediction. Other drivers control hardware. Drivers may also communicate with other drivers; this is called snooping. The INDI architecture places no restrictions on what a driver can do. The only requirement is that it respond to INDI messages that arrive on its stdin stream for its *device* and that it generate valid INDI messages from its *device* on its stdout stream. INDI drivers are most easily written in C using the library functions provided with this system; type man indidevapi for details. The source code for all drivers are included with this package and serve as excellent examples of well written drivers.

Clients, like drivers, may do anything they wish so long as they communicate valid INDI messages over the socket with which they connect to an indiserver. Otherwise clients can be GUIs, command line programs, daemons or other process roles and may be written in any desired language. The sample clients provided in this package are the GUIs ObsCon and ObsCam written in Java and the command line clients get/set/evalINDI written in C. The latter may be used directly but are generally intended to be used by scripts written in perl, python or shell as a handy means to communicate with an indiserver without the need to write socket and XML processing code. The source code for all of these clients is included with this package and serve as excellent examples of well written clients.

Using a TCP socket for clients to connect to an indiserver provides great flexibility. The client and indiserver may be on the same host in which case the simple localhost alias provides a very easy connection. If the clients are on other machines, there are two choices depending on the need for security. By default indiserver listens to port 7624. If the firewall on its host has this port open, then clients on other hosts may connect directly by simply specifying this port when they connect. But if such cavalier connections are deemed unwise, then a secure connection can be made using ssh tunneling. Ssh has the ability to build a secure connection to a remote host in such a way as it appears as a local socket server but in fact transfers this connection to a server on a remote host. It can only do this if an ssh login is available from the client host to the indiserver host. See the -L option in the ssh man page on linux or the ssh tab on the Windows client such as putty. Ssh tunneling thus addresses both access control and secure communications. Using ssh is not necessary when using the GUI clients included with this package because they have the ability to make ssh tunnels already built in (see the -t option on ObsCon and ObsCam).

2 GUI Tools

There are two programs that provide a graphical user interface to the observatory control system. One is ObsCon: an abbreviation for Observatory Control. The other is ObsCam: an abbreviation for Observatory Camera. Both are Java client programs that connect to the INDI network. Multiple simultaneous instances of both these tools may be run at the same time, and all have equal peer control over the system, so take care to arrange an arbitration scheme in a separate manner to determine who has responsibility for operating the equipment and who is just monitoring.

2.1 ObsCon

ObsCon provides command and monitoring capability for all observatory systems except the camera (to operate the camera, see ObsCam in §2.2). In order to function, it must connect to the observatory INDI server. ObsCon is written in Java and distributed as a jar file. It is recommended to run obscon using the convenient script provided, obscon, which sets up a default environment and runs the Java runtime giving it the jar file and any additional arguments.

2.1.1 Command line arguments

ObsCon support the following command line arguments:

-e *n*	specify number of samples in the Environment window graph.
-h *h*	specify direct socket connection to INDI host *h,* default localhost
-i	display inbound INDI messages for debugging
-o	display outbound INDI messages for debugging
-p *p*	specify direct socket port *p,* default 7624
-s	display in a smaller GUI format. This makes obscon use smaller fonts and smaller gaps between GUI components, designed for use on laptops or other small screens.
-t *h s i l*	create ssh tunnel to INDI host *h*, ssh port *s* (default 22), INDI port *i* (default 7624) and login account *l*. This is the default connection mode if none of -t, -h or -p are given. This is used to access an INDI server that is behind a firewall by creating a secure ssh tunnel. It is necessary to have an account on the INDI server and be able to log into that account from outside the firewall using ssh.
-w	ignore and don't save window information. Without this option, obscon will save and restore the location, size and whether it was visible for each obscon window each time it is exited and started. It saves this information to a file named .obscon (note the leading period) in the user's home directory.

2.1.2 Status Indicators

ObsCon makes extensive use of small colored dots to indicate specific state information. These are always one of four colors, as defined in Illustration 2.

- Gray: Idle or unknown
- Green: OK or ready
- Yellow: Busy or in progress
- Red: Alert or problem

Illustration 2: ObsCon status color meanings

2.1.3 Main window

When ObsCon is started, the main window appears, see Illustration 3. Across the top are buttons to open additional windows as described in the following sections.

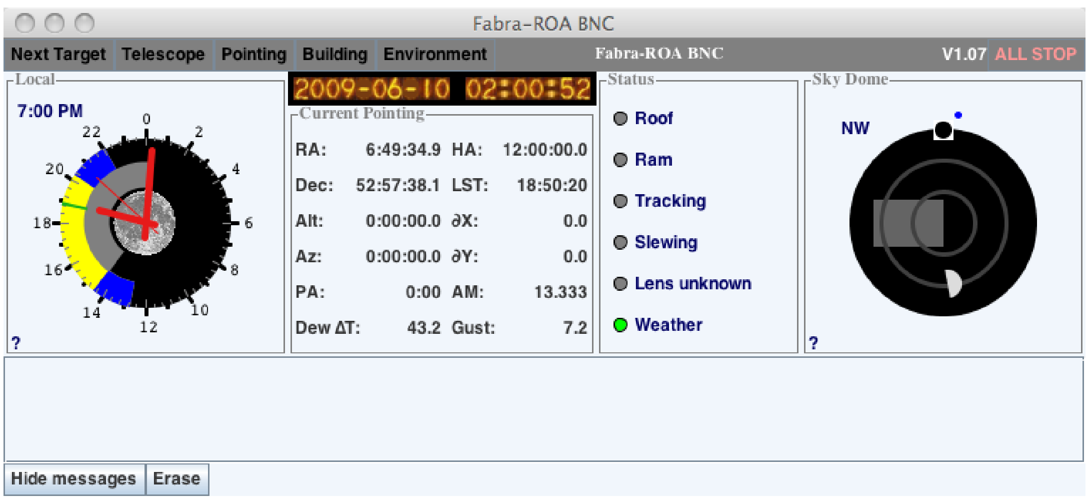

Illustration 3: Main ObsCon window

Across the top are buttons to display the several supporting windows provided by ObsCon.

The left section labeled **Local** contains a 24-hour clock in local time. The yellow and blue ring shows when the sun is above the horizon and the times of dawn and dusk. The gray ring shows when the moon is above the horizon. The time marked in green is the local sidereal time. The local time is displayed in the upper left. Clicking on the question mark (**?**) displays a dialog with this information in more quantitative terms.

The section labeled **Current Pointing** shows information about where the telescope is pointing and critical environmental information (dewing and wind gust). The current UTC is displayed above this section.

The section labeled **Status** shows the most important observatory system information such as telescope, roof, mirror cover and weather alerts.

The section labeled **Sky Dome** shows a sketch of the observatory showing whether the roof and ram arm open, the current wind direction, current telescope pointing direction and symbols showing the sun and moon if they are above the horizon.

The bottom section of the window shows messages from the various INDI drivers. These may be hidden, erased and scrolled with the controls provided.

2.1.4 Next Target

This window can display current and daily planning sky location information for potential viewing targets and can create or save observing lists of planned targets. See Illustration 4.

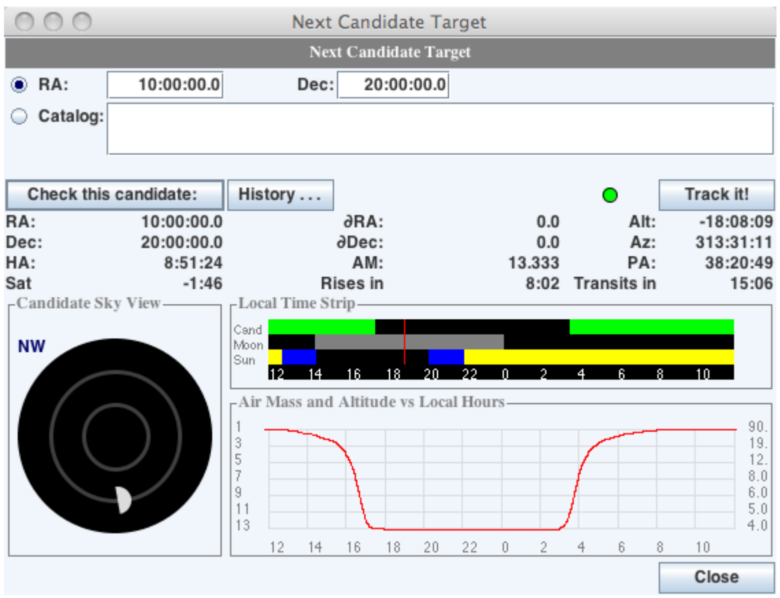

Illustration 4: ObsCon Next Candidate window

Across the top are fields to define a target of interest using either of two methods. If the first method is chosen, enter the **RA** and **Dec**lination for a fixed target at Epoch J2000. If the second method is chosen, enter the name of a catalog[2] entry or the definition of a target in either edb[3] or Two-Line element format.

After a target is defined, click **Check this candidate**. This will display current information, a time strip showing when the target and Sun and Moon are up today and a graph of altitude and airmass. The strip and graph span one day centered on local midnight. If the candidate is currently above the horizon, it will also be displayed in the sky map at the bottom.

Clicking **History** will bring up a table of candidates checked so far during this session. A candidate may be checked again by double-clicking or selecting it and clicking **Check**. A selected entry may be removed from the list by clicking **Delete**. The list may be saved by clicking **Save** and a previously saved list may be read by clicking **Browse**.

If it is desired to track the current candidate, click on **Track it!** This will slew the telescope to the target and begin to track it.

---

2 Catalogs are stored in /usr/local/octavi/catalogs.

3 http://www.clearskyinstitute.com/xephem/help/xephem.html#mozTocId468501

2.1.5 Telescope

The Telescope window allows direct control over the telescope, the mirror cover and camera focus. Refer to Illustration 5.

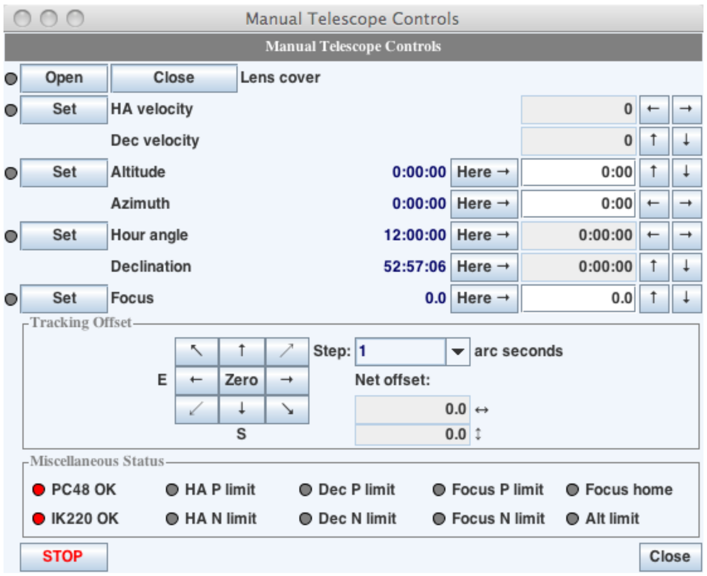

Illustration 5: Manual Telescope Control window

The top row has buttons to open and close the **Lens cover**.

The next two rows allow entering a desired slew velocity for each axis. Enter the desired values in degrees/second then click **Set**. The arrows provide a convenient means to increase or decrease the value by 0.1 degrees per second.

The next two rows show the current telescope pointing direction in an Altitude and Azimuth coordinate system. New values may be entered as desired then clicking **Set** will slew the telescope to the given direction and stop. Click **Here** to copy the current value to the new values field. Use the arrows to change the values in increments of 10 degrees.

The next rows are similar, but allow entering a desired **HA** and **Dec** pointing direction. Enter the desired pointing direction then click **Set**.

The next row displays the current focuser position and may be changed to a new position by entering a value into the given field and clicking **Set**. As with pointing direction, shortcuts are provided for copying the current position into the desired field and incrementing the position.

The section labeled **Tracking Offset** offers a means for injecting modest offsets into the telescope pointing position. The offset distance may be chosen from the pulldown menu. Clicking on any of the arrow buttons adds the current offset in the given direction to the total in each dimension. The two text field show, the current net total offset in each principle direction. All offsets in both direction directions may be removed at any time by clicking on **Zero**.

The section labeled **Miscellaneous Status** shows the state of various limit switches and equipment states.

Clicking on **STOP** will immediately bring the telescope to a controlled stop.

2.1.6 Pointing Model

Clicking on Pointing in the ObsCon main window will bring up the Pointing Model window, see Illustration 6. The purpose of a pointing model is to capture a representation of the imperfections of a telescope mount and use this information to point more accurately at sky targets.

Note that the pointing model provided by this system can only compensate for systematic errors, *i.e.*, errors that are stable and repeatable every time for a given sky direction. Errors that do not repeat, such as worm gear wear or phase imperfections, are not modeled with this system. If there is any reason to suspect that any of the model errors have changed, such as removal and replacement of optics, mount adjustments, drive train wear, or accidents then a new model must be created. Some observatories find their models change with season but are otherwise repeatable and so they save and install models for winter, spring, summer and fall.

The model consists of several scaler and trigonometric terms, one for each modeled mechanical error. The terms combine to form a net pointing error at each position in the sky. The errors are added to the positions reported by the axis encoders to form the location actually pointed to in the sky. When used in reverse, the model is applied to the desired sky location and computes the encoder values required to point to that location.

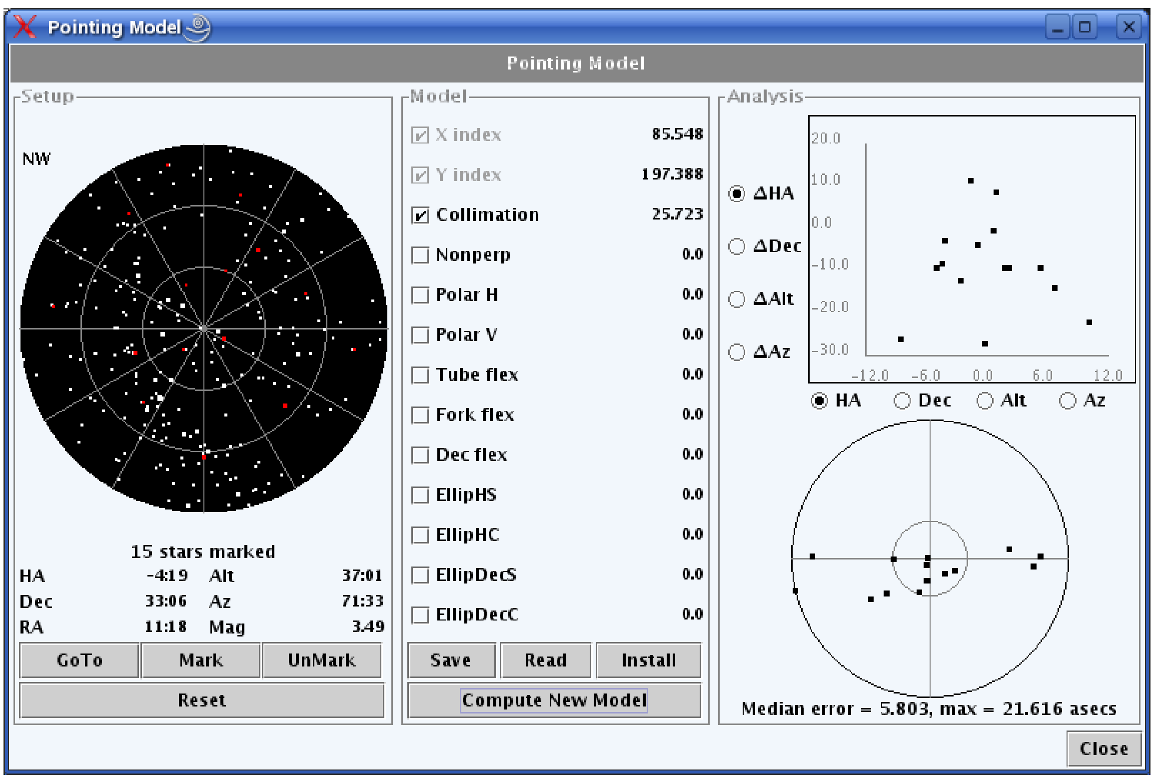

Illustration 6: Pointing model

The Pointing Model window is divided into three sections, left to right. The left section displays a map of the current sky, showing reference stars from a standard catalog[4] and sky locations that are part of a model. The map is updated once per minute to track the movement of stars across the sky. Note that modeled locations do not move over time because they are fixed on the sky.

The center section shows the value of each term of a pointing model in arc seconds, allows selecting terms to be used in a new model and can save new and read previously defined pointing models to files on disk. The

The right section displays two graphs that allow one to analyze the quality of a model. The top cartesian graph can show model errors, in arc seconds, in and against several sky coordinates. The lower polar graph shows the distribution of errors on an HA/Dec polar plot. Clicking on a star in either plot will display its quantitative information, circle it temporarily on all plots and offer the option of removing it from the list of stars used in modeling.

The points on the sky map in the left section are color coded to show their role in a pointing model, as shown in Table 1.

Color	***Meaning***
White	Star from calibration catalog.
Green	Selected to display quantitative information.
Blue	Star will be added if Marked.
Red	Location is used in a pointing model.

Table 1: Colors of stars in Pointing Model sky map

To gather data for a new model, start by clicking on a catalog star which will turn it green and will display its current coordinates in several different frames of reference. Clicking **Goto** will cause the telescope to slew and track to that position using the currently installed model. Use the Offset commands in the ObsCon Telescope window, §2.1.5, to center the star then click **Mark** to add its location to the new model and draw it as red on the map. Repeat this procedure with a good sample of stars around the sky.

To remove the last star from in a new model, click **Undo** as far back as desired. To remove all stars from a candidate model and begin again, click **Reset**.

The column of check boxes and numbers that dominates the center section shows each term of a pointing model and whether that term is being used. It is important to add only enough terms into the model to achieve the desired pointing accuracy. Extra terms that do not add significantly to the model accuracy can make the model less stable between the calibration points.

Once a good collection of stars have been added to a model and the desired terms selected click **Compute New Model**. The values assigned to each selected term will be displayed in the center table. You may repeat with different stars or terms as often as you like. If you wish you may **Save** the model at any time to a disk file. Click **Install** to make this model become the active default model in use by the telescope control system.

Table 2 describes each term available for use in a pointing model[5] in more detail. The columns Δh and Δ show the formulas used by the model for each term to compute the error in hour angle and declination, respectively. The nominal telescope HA and Dec axes are referred to as X and Y, respectively. In the formulas, is the latitude of the observatory.

4 The standard catalog is called pointingstars.edb. It is searched, in order, in the current directroy, in $OBSHOME/config and then in the obscon.jar file.
5 Terms are consistent with those in Tpoint, see http://www.tpsoft.demon.co.uk/pointing.htm

Term	***Description***	**Δ*h***	**Δ**
XIndex	X axis home position	X_0	
YIndex	Y axis home position		Y_0
Collimation	Optical/mechanical misalignment	sec	
Nonperp	Non-perpendicularity of axes	tan	
PolarH	Polar axis azimuthal error	cos h tan	sin h
PolarV	Polar axis altitude error	sin h tan	cos h
TubeFlex	Tube sag	cos sin h sec	cos cos h sin – sin cos
ForkFlex	Fork sag		cos h
DecFlex	Dec sag	cos cos h + sin tan	
EllipHS	Ellipticity of X	sin 2*h	
EllipHC		cos 2*h	
EllipDecS	Ellipticity of Y		sin 2*
EllipDecC			cos 2*

Table 2: Pointing model terms

2.1.7 Building

Clicking on Building in the main ObsCon window will display the building monitoring and control window, as shown in Illustration 7.

The top portion of the window contains buttons and indicators to open and close the roof and end ram.

The center portion shows the current and target temperatures for the air conditioning system.

Next are the controls for the CCD chiller. This allows turning the chiller running On or Off, setting Local (front panel) or Remote control, setting a new set-point temperature and monitoring the current fluid temperature. Note that the communications protocol to the chiller does not allow INDI to determine whether the chiller running is On or Off, nor whether it is operating in Local or Remote control mode. Therefore, the button states shown are only accurate if they were the result of commands issued from obscon and nothing different was performed on the chiller's own front panel. Note also that when the chiller is in Local control mode, it is not possible for obscon to change between running On and Off nor to set a new set-point temperature. For this reason, these controls will be disabled when obscon sets Local mode. Regardless of the control mode, obscon can always correctly display the current set-point temperature and the current fluid temperature.

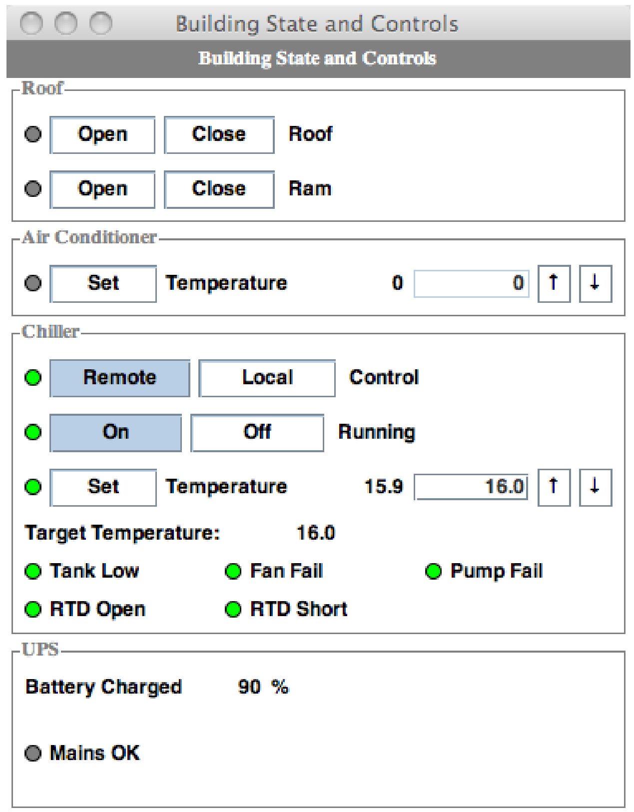

Illustration 7: Building window

The lower portion shows the percentage of battery charge on the UPS and whether the main commercial power is currently available.

2.1.8 Environment

This window shows the current environmental parameters available from the weather station and other sensors in the observatory and can also show graphs of previous values over various time intervals. See Illustration 8.

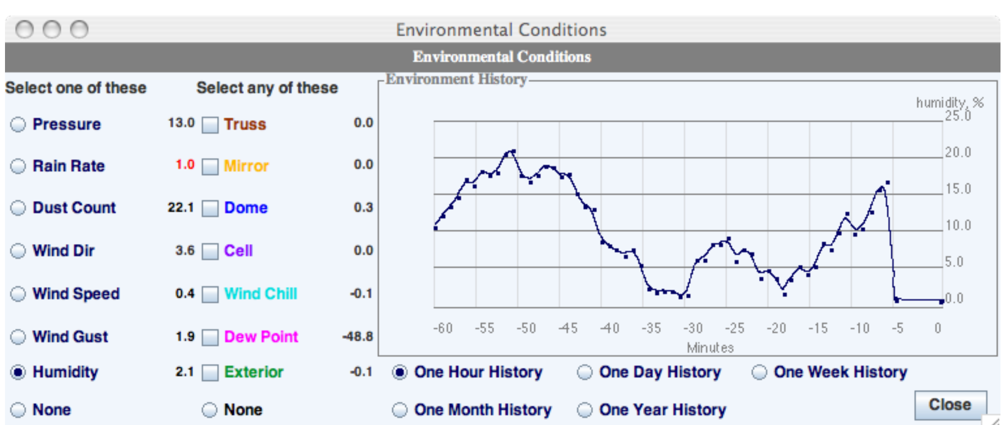

Illustration 8: ObsCon Environment window

Any one of the parameters in the left column may be selected for graphing. The reason only one at a time may be selected is because the units are generally different. The graph will show the scale for these selections on the right side of the graph. Then in addition any number of parameters in the second column may also be selected for graphing. The reason any of these may be selected at one time is because they are all temperatures using the same units. The temperature scale for these values is shown on the left side of the graph.

The time interval for the graph is chosen from the collection of radio boxes below the graph. Intervals from the past hour to the past year may be selected.

Note that quite a lot of data must be sent to build the graphs. Over slow connections, it might be desirable to reduce the amount of data by using the -e command line to obscon (see §2.1) option to reduce the amount of data. Of course, using less data will make a courser graph.

2.1.9 Implementation

The main class is ObsCon. It cracks command line arguments, builds the GUI, makes a connection to the INDI server and waits forever to handle GUI events and INDI messages.

The server connection is built and serviced in ServerIO. The connection can be built using a typical socket, or it can be built via an ssh tunnel. A tunnel requires an ssh login on the target machine. The command line arguments must include the ssh port and login name, then the password will be prompted for interactively. The ssh tunnel is built using SSHTunnel and the ch package in the obsio directory.

Once the connection is built, incoming INDI messages are formatted into an INDIMsg. Each INDIMsg is given to DispatchMsg which runs in a separate thread. The device and name from INDIMsg are combined and used as a hash lookup to find the corresponding subclass of GUIUpdate to run. Thus there is one subclass of GUIUpdate for each possible incoming device/name pair INDI message.

Outgoing INDI messages build an INDIMsg with the appropriate content and are given to ServioIO for transmission.

Each pushbutton in the main ObsCon frame is associated with its corresponding frame and display it when pushed.

2.2 ObsCam

ObsCam stands for Observatory Camera and is the primary means for operating the image detector. It can connect to and control any camera for which there is an INDI driver. It can also read and write FITS files from and to disk. ObsCam is intended only as a basic camera control and image display tool. It is not intended to compete with very elaborate control and processing tools.

2.2.1 Command line arguments

ObsCam supports the following command line arguments:

-h *h*	specify direct socket connection to INDI host *h,* default localhost
-i	display inbound INDI messages for debugging
-o	display outbound INDI messages for debugging
-p *p*	specify direct socket port *p,* default 7624
-s	display in a smaller GUI format. This makes obscam use smaller fonts and smaller gaps between GUI components, designed for use on laptops or other small screens.
-t *h s i l*	create ssh tunnel to INDI host *h*, ssh port *s* (default 22), INDI port *i* (default 7624) and login account *l*. This is the default connection mode if none of -t, -h or -p are given. This is used to access an INDI server that is behind a firewall by creating a secure ssh tunnel. It is necessary to have an account on the INDI server and be able to log into that account from outside the firewall using ssh.

2.2.2 Main window

The ObsCam window is shown in Illustration 9.

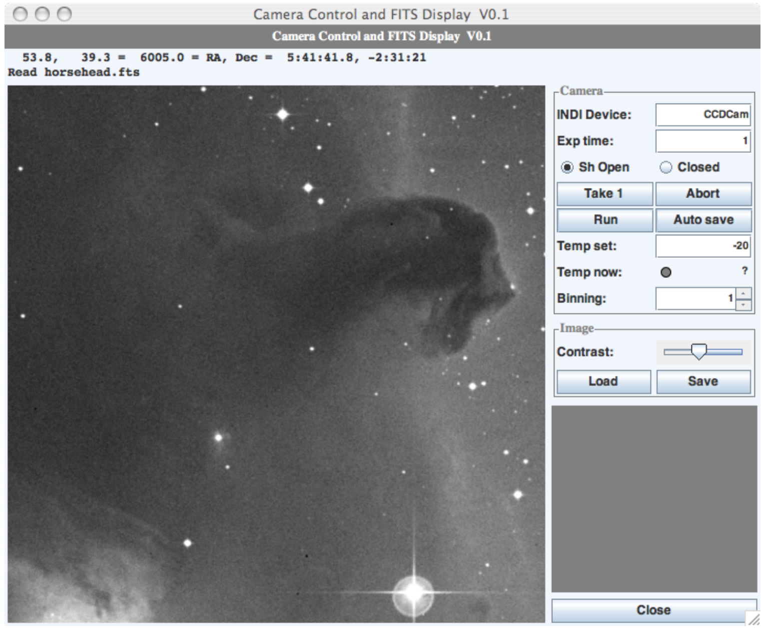

Illustration 9: ObsCam window

The obscam window is dominated by the image display rectangle in the lower left. The area always shows the entire image and unused portions are shown in blue. Moving the mouse over the window will show a magnified view in the lower right rectangle.

Before taking an image, set the exposure time in the text field labeled **Exp time**. The units are in seconds. Also set the shutter to **Sh Open** or **Closed** as desired for normal exposures or bias or dark calibration frames. To take one exposure, click **Take 1**. When the image is complete, it will be shown in the image display rectangle. To automatically take images one after another, toggle **Run** on. To stop after the next image, toggle **Run** off. Click **Abort** to abandon an exposure before it is complete started by either method. To save each image as it arrives to a disk file, click **Auto Save** on. The name will consist of the Date and Time. All files are stored as 16 bit FITS[6] files.

The camera cooler target temperature is set by entering the desired value in the text field next to **Temp set** then typing Enter. The current cooler temperature is displayed next to **Temp now**, the status light being green if the cooler is at the target temperature, yellow if it is moving towards the target temperature, red if there is a cooler error or gray if the cooler is off. Pixels may be binned (equally horizontally and vertically) using the spin box next to **Binning**.

The image **Contrast** can be set using the slider. The image can be saved by clicking **Save**. An existing FITS files can be loaded for viewing by clicking **Load**.

Each time an image is loaded (either from the camera or a disk file) and secondary window is opened in which are displayed the FITS header fields. See Illustration 10. This window may be moved and resized as desired but will always appear.

6 http://heasarc.gsfc.nasa.gov/docs/heasarc/fits.html

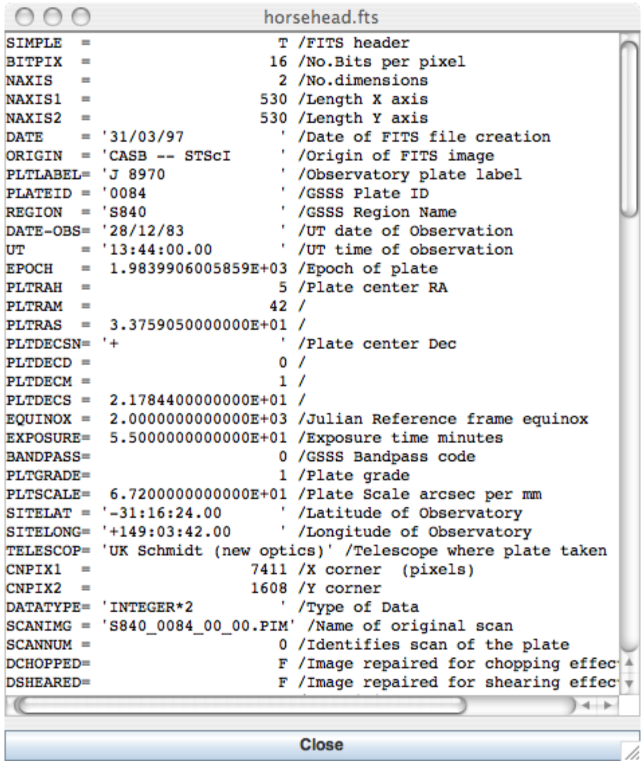

Illustration 10: ObsCam FITS header window

2.3 QExCon

QExCon stands for Queued Execution Control. QExCon is offers the means to operate the observatory in a completely automated fashion. Using QExCon you define the INDI commands you want to execute, define the target and any additional constraints for the observation, then the QExCon device driver will decide the best time to perform the request. Many requests may be pending simultaneously and the QExCon driver will always attempt to perform each of them at the best possible time.

2.3.1 Command line arguments

QExCon supports the following command line arguments:

-h *h*	specify direct socket connection to INDI host *h,* default localhost
-i	display inbound INDI messages for debugging
-o	display outbound INDI messages for debugging
-p *p*	specify direct socket port *p,* default 7624
-t *h s i l*	create ssh tunnel to INDI host *h*, ssh port *s* (default 22), INDI port *i* (default 7624) and login account *l*. This is the default connection mode if none of -t, -h or -p are given. This is used to access an INDI server that is behind a firewall by creating a secure ssh tunnel. It is necessary to have an account on the INDI server and be able to log into that account from outside the firewall using ssh.

2.3.2 Main window

The QExCon main window consists of two tabs. One is used to Build a new scheduled observing request. The other is to View all existing requests. Each tab is divided into two panes. Each pane may be individually scrolled as necessary to view its contents.

2.3.3 Build tab

The Build tab of QExCon is shown in Illustration 11.

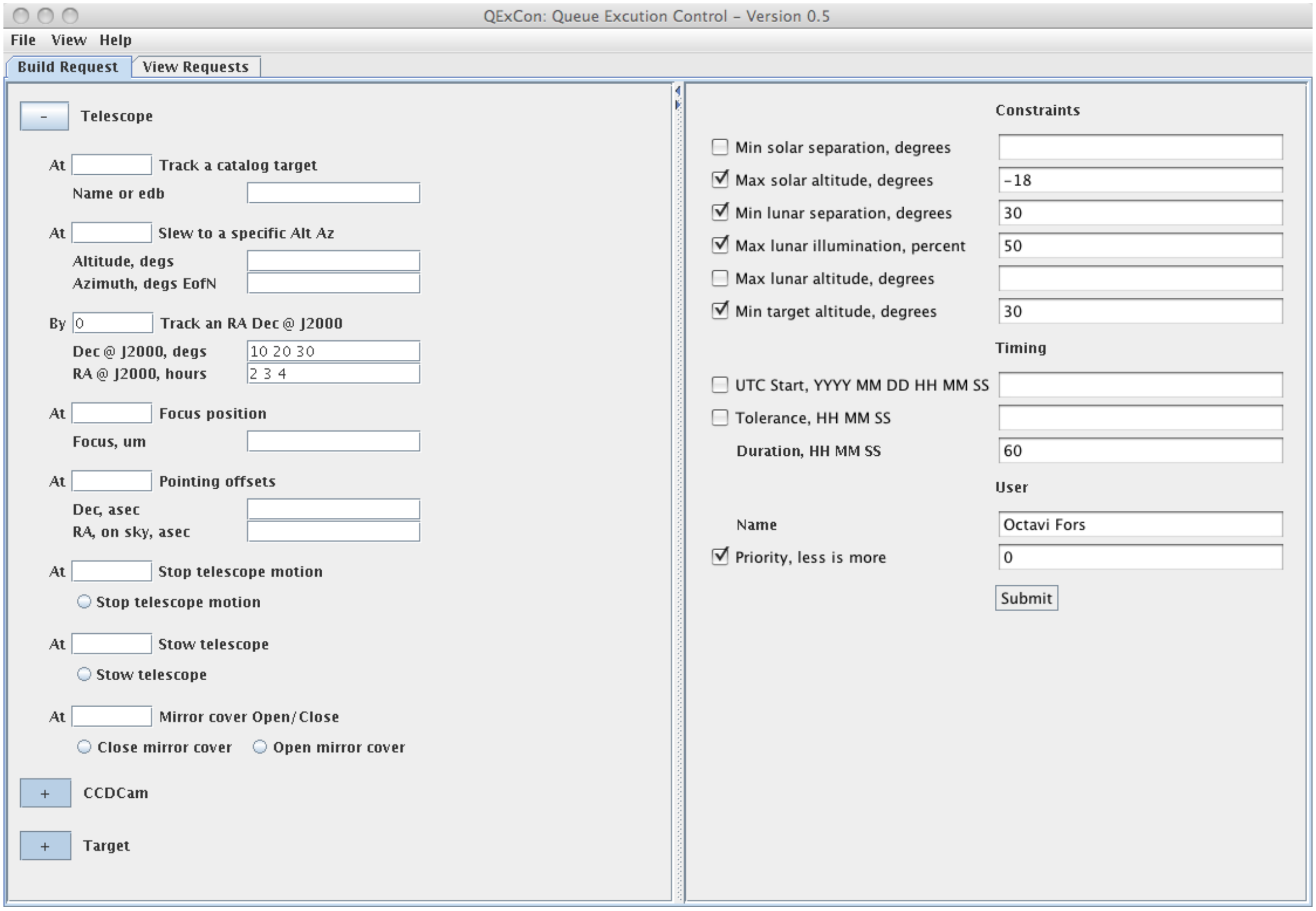

Illustration 11: QExCon Build tab

The left pane of the Build tab displays all INDI properties on the system. This is not a live display, such as you might see in ObsCon, but is simply a static list of each property name, including each element if it is an array. The idea of the QExCon system is to collect any set of INDI commands and trigger them at some time in the future to perform some actions.

Beside each property is a text entry field labeled with "At" or "By". These fields are for entering a time difference with respect to the moment when this request is scheduled to be performed. If a property requires some finite time to accomplish, such as slew to a target or open a roof, then the label is "By". If the property event happens essentially instantly, such as closing a camera shutter, then the label is "At". For example, if a property such as 1-Wire.Roof.Open, which will be labeled "By" because it requires some time to perform, is set to 30, it means to issue that property command such that it will complete 30 seconds before the scheduled time for the request.

Also beside each property is a checkbox labeled "Abort". All such commands are executed immediately if the request is aborted before it finishes normally. This can happen if Canceled from the GUI or if a higher priority request is chosen by the scheduler to begin before the current Running request completes.

The right pane of the Build tab lists a set of constraints. When the QExCon control system chooses a time to run this request, it will attempt to satisfy all of the constraints that are checked. Turning on more requests gives you more control of the observing circumstances for this particular request, but turning on fewer constraints will give the scheduler more flexibility to compare this request with all others and find a solution that is suitable for more requests over all. In addition to these constraints, QExCon will always attempt to choose a time that places the target as high in the sky as possible.

Fields that require date and time are in UTC and must use format YYYY MM DD HH MM SS. Fields that require angles may use either the format DD MM SS or decimal degrees. For all time and angle fields, in addition to a space the separator may also be any non-digit character such as slash "/", hyphen "–" or colon ":". Tolerance and Duration time fields are in decimal seconds.

Specifying UTC Start instructs the scheduler to attempt executing the request at that time. If other constraints are also checked then the scheduler will move away from the Start time up to the specified Tolerance as necessary in order to meet all constraints and avoid other requests already scheduled.

If the After or Before constraints are specified, then under no circumstances will the request be executed before or after the specified time, respectively.

In addition to timing and astrophysical constraints, you may also assign a numeric priority to the request. These have no physical meaning but are simply used by the scheduler to give first consideration to requests with higher priority. It is expected that a group of users will agree upon some strategy for a range of values to use. Any real values may be used but always numerically smaller values have greater priority than larger values. Note that the priority system effectively becomes meaningless if all requests are assigned the same priority value.

A field is also provided to enter a name for the observation. This is simply recorded and carried along with the observation request as a helpful convenience, it does not play an active role in the scheduling algorithm.

Once the properties to be performed have been set in the left pane and all desired constraints are specified in the right pane, click **Submit** to enter the request into the QExCon queue. If there are any errors, they will be shown otherwise the new request is stored in the QExCon database.

2.3.4 View tab

The View tab is used to inspect and manage the current contents of the QExCon observing queue. See Illustration 12.

The left pane shows a summary of each request, one per line. The information listed is the time it has been, or was, scheduled, when it was submitted, its current state and the User name. The possible states are shown in Table 1.

Request State	***Meaning***
New	Has not yet been scheduled
Scheduled	Scheduled but not yet executed
Running	Currently being executed
Finished	Completed successfully
Canceled	This request was cancelled
Failed	Execution failed for some reason

Table 1: QExCon request states

Clicking on any request in the left panel will display the full details of that request in the right panel. Here is shown everything known about the request. Also shown is a history of each step that occurred in the life time of the request, including any information about why it may have failed or been rejected.

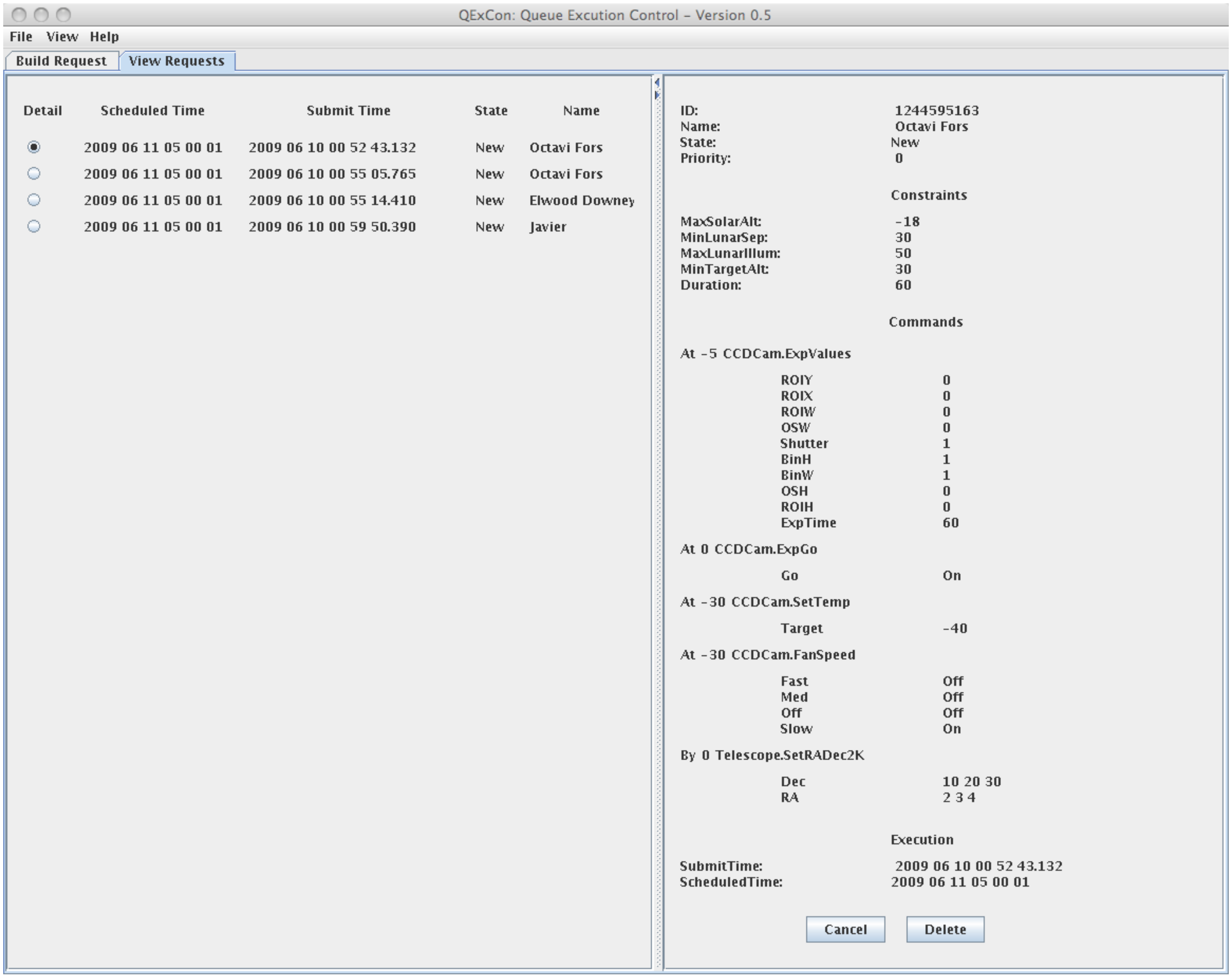

Illustration 12: QExCon View tab

Also on the right panel are buttons to permanently **Delete** the request from the QExCon database and, if it has not yet been fully executed, to **Cancel** the request.

2.3.5 Scheduling Algorithm

Each time the scheduler needs to decide the next request to execute the indiqex process scans all requests in the database that are in a state of New or Scheduled. The request that ends up sorted soonest in the future is then executed when its time occurs. The sorting procedure is performed each time the queue is modified, after each execution completes or after each hour of idle time. By resorting on a frequent basis, the system accommodates requests that may have been added, cancelled or deleted and responds to temporal effects such as the current temperature and humidity (Although not included in this implementation, other temporal effects such as clouds or seeing could also be accommodated in principle).

The first step of the algorithm is to change all New or Scheduled requests which specify a Start time plus Tolerance or a Before time that is already in the past to state Failed. Next all remaining requests in state New or Scheduled are sorted by decreasing priority. All requests at the same highest priority are assigned as described next, then requests at the next lower priority and so on. Requests that do not specify a priority explicitly are sorted last. Note that if all requests have the same Priority then effectively there is no priority effect.

Each request contains a Constraints table. This is a table consisting of 1,440 boolean elements or “slots”, where each slot corresponds to each minute in the next 24 hours. Each slot in the table is set to True or False depending on whether *all* specified constraints for that target are satisfied at that moment, including consideration of After, Before and Start and Tolerance times, if specified.

Each request also stores the total number of slots that are True in its Constraints table, the Duration in terms of the number of contiguous slots required, and the index of a Preferred slot that indicates the moment at which the scheduling algorithm will begin its search. If the Start constraint in the request is specified then the Preferred slot is simply set from this value. Otherwise, the request must define an astronomical target, that is, one that is defined using an RA and Dec or using orbital elements, in which case the Preferred slot is set to correspond to the slot at the moment the target is at its highest apparent horizon altitude, subject to all other specified constraints. The request is marked as Failed if it contains neither Start nor an astronomical definition.

There is also one Master assignment table. This is a table consisting of 1,440 pointers to requests, where again each pointer corresponds to each minute in the next 24 hours. At the start of the algorithm, all entries are set to Null. As the scheduling algorithm progresses, this table is gradually filled in to point to the request scheduled for each time slot as the requests are assigned. Contiguous entries are assigned to the same request for its Duration.

The next step, then, is to compute the information about the target at each slot time, setting the Constraints table accordingly and determining the number of True slots and the Preferred slot along the way.

Next, the set of eligible requests is sorted in order of increasing number of True slots in its Constraints table. In this way requests which are more tightly constrained in time are considered before requests that may be observed over a wider range of times.

Finally the algorithm is ready to make assignments. Each request is first attempted to be scheduled at the Preferred time slot in the Master assignment table. If the Preferred time slot is already assigned then the surrounding slots before and after are checked in ever expanding moments away from the Preferred time slot. The request is scheduled at the time which corresponds to the first slot that qualifies. In order for a slot to qualify, it and sufficient subsequent contiguous slots to provide for the Duration of the request must be Null in the Master assigned table. If such a region is found, a pointer to the request is set in each slot in the Master assigned table to mark those slots as being unavailable for further assignment and the request state is set to Scheduled.

The algorithm repeats in this way until all requests are either assigned, in which case their state is set to Scheduled or no assignment was possible in which case they are assigned state New to be attempted next time.

If a running request is canceled, no special processing is performed other than to mark its state as Canceled and move on to the next request. If special processing is required, for example to stop the telescope or close the camera shutter, this must be performed by a suitable INDI client or script. In particular, clients that wish to present urgent Targets Of Opportunity for execution should first cancel any currently running request, perform any necessary equipment cleanup actions, then submit the TOO request for scheduling.

2.3.6 Qex.cfg

The QExCon driver requires a configuration file to be set up. The name is qex.cfg and is in the usual directory, /usr/local/octavi/config. This file defines the following variables.

qpath	Specifies the full path of the directory containing the qexcon data base. The database simply consists of one file per request. The name of the file is the request ID number followed by the extension *.qex*. The contents of the file are the xml plain text description of the request. The file is updated whenever its contents change, such as to the change the state.
bpath	Specifies the full path of the directory in which any BLOBs received during the execution of a request for the property named by **PropSave** (see below) are stored. Thus, this is typically where image files will be stored.
PropRA	Name of the INDI property that specifies the RA @ J2000 of a fixed astronomical target. The format is Device.Name.Element. For example, on this system it should be Telescope.SetRADec2K.RA.
PropDec	Name of the INDI property that specifies the Dec @ J2000 of a fixed astronomical target. The format is Device.Name.Element. For example, on this system it should be Telescope.SetRADec2K.Dec.
PropEDB	Name of the INDI property that specifies an astronomical target using an edb or TLE specification. The format is Device.Name.Element. For example, on this system it should be Telescope.SetCatalog.entry.
PropSave	Name of the INDI property that will contain a BLOB of data as a consequence of executing a request. Typically it is the property that contains pixels from a camera or other instrument. The contents of the BLOB are stored in the directory specified by the **bpath** variable (see above). The format is Device.Name.Element. For example, on this system it should be CCDCam.Pixels.Img.
port	TCP/IP port number used to connect to the INDI server. The default value is the standard INDI port 7624.
host	TCP/IP host name or IP address used to connect to the INDI server. The default value is *localhost*.

2.3.7 XML Database Format

Each QExCon database entry is stored in XML format. We describe the format of a request by way of the example below. Each individual observation request is contained in an element named INDIObservation. When it is necessary to collect one or more of these into a list, the outermost list element is named INDISchedules.

Within each INDIObservation are the following subelements.

ID	This element contains is a unique number assigned to each observing request. The number is not intended to have any meaning but it can be said it is based on the time when the request was submitted.

Constraints	This element contains subelements each of which define the possible constraints to be applied when scheduling this observation. If a given constraint is not specified for this request, its element may be absent.
User	This element contains subelements that contain an identifier for the person or organization that submitted the request and a priority. If a priority is not specified for this request, this element may be absent.
INDICommands	This element contains a collection of At or By subelements. These in turn contain the exact INDINew* commands to be issued when this observation is executed. It also contains a subelement named Abort. This is the set of INDI commands that are to be sent if a request that is underway is to be aborted before it completes.
Execution	This element contains information about when and how the request is being executed. The state element indicates the current mode of the request. Other subelements capture when the states will or did change. This element may also include any number of History sublements each of which contains a brief description of some event that occurred during the lifetime of this request. Each history element contains a time attribute to record the moment when the event occurred.

All references to an absolute date and time use UTC in the format YYYY MM DD HH MM SS.

```
<INDISchedules>
      <INDIObservation>
            <ID>334376768</ID>
            <Constraints>
                  <MinSolarSep>10</MinSolarSep>
                  <MaxSolarAlt>-18</MaxSolarAlt>
                  <MinLunarSep>30</MinLunarSep>
                  <MaxLunarIllum>50</MaxLunarIllum>
                  <MaxLunarAlt>10</MaxLunarAlt>
                  <MinTargetAlt>30</MinTargetAlt>
                  <SatIsSunLit>0</SatIsSunLit>
                  <UTCStart> 2009 05 13 13 45 00 </UTCStart>
                  <Tolerance> 1 0 0 <Tolerance>
                  <UTCAfter> 2009 05 10 00 00 00 </UTCAfter>
                  <UTCBefore> 2009 05 15 00 00 00 </UTCAfter>
                  <Duration> 0 1 0 <Duration>
            </Constraints>
            <User>
                  <Name>NGC 1332</Name>
                  <Priority> <Priority>
            </User>
            <INDICommands>
                  <at t=”0”>
                      <newSwitchVector device=”CCDCam” name=”ExpGo”>
                         <oneSwitch name=”Go”>On</oneSwitch>
                      </newSwitchVector>
                  </at>
                  <at t=”-1”>
                      <newNumberVector device=”CCDCam” name=”ExpValues”>
                         <oneNumber name=”ExpTime”>30</oneNumber>
                         <oneNumber name=”ROIW”>0</oneNumber>
                         <oneNumber name=”ROIH”>0</oneNumber>
                         <oneNumber name=”OSW”>0</oneNumber>
                         <oneNumber name=”OSH”>0</oneNumber>
                         <oneNumber name=”BinW”>1</oneNumber>
                         <oneNumber name=”BinH”>1</oneNumber>
                         <oneNumber name=”ROIX”>0</oneNumber>
                         <oneNumber name=”ROIY”>0</oneNumber>
                         <oneNumber name=”Shutter”>1</oneNumber>
                      </newNumber>
                  </at>
                  <by t=”0”>
                      <newTextVector device=”Telescope” name=”SetCatalog”>
                         <oneText name=”entry”>Mars,P</oneText>
                      </newTextVector>
                  </by>
                        . . .
            </INDICommands>
            <Execution>
                  <State> one of: New, Scheduled, Running, Finished, Cancelled, Failed
            </State>
                  <SubmitTime> 2009 05 12 02 05 16 </SubmitTime>
                  <ScheduledTime> 2009 05 13 11 34 00 </ScheduledTime>
                  <StartTime> 2009 05 13 11 33 00 </StartTime>
                  <EndTime> 2009 05 13 11 33 50 </EndTime>
                  <History time=”2009 05 13 11 33 00”>Start Observation</History>
                         . . .
            </Execution>
      </INDIObservation>
            . . .
</INDISchedules>
```

3 INDI Properties

This section lists all INDI Devices and their Properties. There is one table per Device. The table lists the name, type and permission for each Parameter. In addition, Number parameters list the name and units for each Element; Switch parameters list the rule and name of each switch Element; and Text parameters list the name and purpose of each Element.

After each parameter is a brief description and the meaning of the four standard INDI States as they apply to that parameter. Drivers send their parameters with defXXX messages or when they change unless otherwise noted.

Following the tables is a diagram showing inter-driver communication channels.

3.1 Telescope

Telescope – telescope control				
Name	***Type***	***Perm***	***Element***	***Units***
Pointing	Number	RO	RA2K	hours@J2k
			Dec2K	degs@J2k
			RAEOD	hours@EOD
			DecEOD	degs@EOD
			HA	hours +W
			Alt	degs up
			Az	degs E of N
			AM	air mass
			PA	PA, degs
			XVEL	HA vel, °/s
			YVEL	Dec vel, °/s
			Focus	nm
			JD	Time of data
Broadcast current telescope pointing information at approximately 2 Hz. Idle=no motion OK=tracking Busy=slewing Alert=fault condition				
Focus	Number	WO	position	nm
Command a new Focus position. Idle=no motion OK=in position Busy=moving Alert=fault condition				
SetCatalog	Text	WO	entry	name or edb
Command telescope to track the given catalog or edb format specification. Idle=no motion OK=in position Busy=moving Alert=not found or bad format.				
SetAltAz	Number	WO	Alt	degs up
			Az	degs E of N
Command telescope to slew to the given alt/az. Idle=no motion OK=in position Busy=moving Alert=fault condition				
SetHADec	Number	WO	HA	Hours
			Dec	degrees
Command telescope to slew to the given HA/Dec. Idle=no motion OK=in position Busy=moving Alert=fault condition				
SetVelocity	Number	RW	HA	degs/sec west
			Dec	degs/sec north

Telescope – telescope control				
Name	***Type***	***Perm***	***Element***	***Units***
Command telescope to slew at the given velocities. Idle=no motion OK=at velocity Busy=moving Alert=fault condition				
SetRADec2K	Number	WO	RA	hours
			Dec	degs
Command telescope to track the given RA/Dec coordinates at epoch J2000. Idle=no motion OK=in position Busy=moving Alert=fault condition				
Stow	Switch	WO	Go	AtMostOne
Command move to standard stow position. Idle=no motion OK=in position Busy=moving Alert=fault condition				
Status	Light	RO	PC48OK	PC48 OK
			IK220OK	IK220 OK
			EStop	Emergency stop
			HAPLim	HA pos limit
			HANLim	HA neg limit
			DecPLim	Dec pos limit
			DecNLim	Dec neg limit
			FocusHomeOK	Focus home OK
			FocusPLim	Focus pos limit
			FocusNLim	Focus neg limit
Collection of misc telescope system status indicators. Property status matches highest level of all constituents.				
Stop	Switch	WO	Stop	AtMostOne
Command an immediate halt to all telescope motion. Idle=no effect OK=stopped Busy=stopping Alert=fault condition				
Offsets	Number	WO	RA	arcseconds
			Dec	arcseconds
Inject offsets to the current target position. Idle=unknown OK=in effect Busy=taking effect Alert=fault condition				
PtgModel	Number	RW	Mode	*
			XIndex	
			YIndex	
			Collimation	
			Nonperp	

Telescope – telescope control				
Name	***Type***	***Perm***	***Element***	***Units***
			PolarH	
			PolarV	
			TubeFlex	
			ForkFlex	
			DecFlex	
			EllipHS	
			EllipHC	
			EllipDecS	
			EllipDecC	
*Mode 0: Compute and return a new model that uses only these non-zero fields. *Mode 1: install this as the running model and make it the new default. *Mode 2: install this model as the current candidate. Idle=unknown OK=accepted Busy=in progress Alert=rejected				
PtgStar	Number	RW	Count	N..1 or
			RA2K	hours
			Dec2K	degrees
			HA	hours
			Dec	degrees
			X	hours
			Y	degrees
			Alt	degrees
			Az	degrees
			delHA	arcseconds
			delDec	arcseconds
			delAlt	arcseconds
			delAz	arcseconds
One star of a set used in a model. N stars are sent sequentially, decrementing Count from N to 1. 0 means set is empty. idle=unknown OK=ready Busy=in progress Alert=fault				
PtgMark	Switch	WO	Mark	AtMostOne
			Undo	
			Reset	
Mark (from last SetRADec), Undo last or Reset all stars in model collection. Idle=unknown OK=accepted Busy=in progress Alert=error.				

3.2 Target

Target – candidate target predictions				
Name	***Type***	***Perm***	***Element***	***Units***
AskRADec	Number	WO	RA	hours
			Dec	degs
			epoch	year or 0=EOD
			WantRTS	0 or 1
			WantDesc	0 or 1
			JD	@ JD or 0=now
			ID	unique ID
Ask for local info about a target defined by RA/Dec at a given JD. Causes one Info response with matching ID. Info.Rise/Transet/Set will be filled in unless WantRTS is 0. Causes one InfoDesc response with matching ID unless WantDesc is 0. Idle=void OK=valid query Busy=looking Alert=invalid query				
AskCatalog	Text	WO	name	name*
			WantRTS	0 or 1
			WantDesc	0 or 1
			JD	@ JD or 0=now
			ID	unique ID
Ask for local info about a target defined by name* at a given JD. Causes one Info response with matching ID. Info.Rise/Transet/Set will be filled in unless WantRTS is 0. Causes one InfoDesc response with matching ID unless WantDesc is 0. *name can be any catalog entry, major planets and moons or XEphem edb specification. Idle=void OK=valid query Busy=looking Alert=invalid query				
Info	Number	RO	RA2K	hours@J2k
			Dec2K	degs@J2k
			RAEOD	hours@EOD
			DecEOD	degs@EOD
			PMRA	“/min on sky
			PMDec	“/min on sky
			HA	hours +W
			PA	paral ang, °+W
			Alt	degs up
			Az	degs E of N
			AM	air mass
			Rise[1]	JD*

Target – candidate target predictions				
Name	***Type***	***Perm***	***Element***	***Units***
			Transit[2]	JD*
			Set[3]	JD*
			JD	JD of info
			ID	matching ID
Quantitative response to a previous AskRADec or AskCatalog query. ID matches that in request. Rise/Transit/Set will be 0 unless query set WantRTS. Idle=void OK=valid query Busy=working Alert=invalid query; * -1 = never up, -2 = circumpolar; [1]previous rise if up now else next; [2]next transit after Rise; [3]next set if up now else previous				
InfoDesc	Text	RO	Description	prose
			ID	matching ID
Prose description response to a previous AskRADec or AskCatalog query. ID matches that in request. Idle=void OK=valid query Busy=looking Alert=invalid query				

3.3 Environment

Environment – local current and historical conditions				
Name	***Type***	***Perm***	***Element***	***Units***
Now	Number	RO	AirTemp	°C
			DewPoint	°C
			WindChill	°C
			AirPressure	hPa
			Humidity	%
			WindDir	degs E of N
			WindSpeed	mps
			WindGust	mps
			RainAccum	YTD mm
			RainDetected	0 or 1
Broadcast current environmental stats every five seconds. Idle=not available OK=valid and in safe range Busy=updating Alert=out of range				
AskEnv	Number	WO	JD	@ JD or 0=now
			ID	unique ID
Ask for environmental stats from logs nearest to the given JD. Causes one JDEnv response with matching ID. Idle=void OK=valid query Busy=looking Alert=invalid query				
JDEnv	Number	RO	AirTemp	°C
			DewPoint	°C
			WindChill	°C
			AirPressure	hPa
			Humidity	%
			WindDir	degs E of N
			WindSpeed	mps
			WindGust	mps
			RainAccum	YTD mm
			RainDetected	0 or 1
			JD	JD
			ID	matching ID
Historical environmental stats for the given JD. Caused by AskEnv with matching ID. Idle=void OK=valid query Busy=looking Alert=invalid query				

Environment – local current and historical conditions				
Name	***Type***	***Perm***	***Element***	***Units***
WAOverride	Switch	WO	Override	AtMostOne
May be used by clients to implement a Now "Weather Alert Override". The only functionality provided by the driver is to time out back to Off after a configurable time period. Idle=no "Override" OK=(unused) Busy=updating Alert="Override" in effect				
Limits	Number	RO	MaxHumidity	%
			MaxWindSpeed	mps
Report maximum safe operating conditions. Sent once per successful connection. Idle=void OK=valid query Busy=looking Alert=unavailable				

3.4 Time

Time – current observatory time and circumstances				
Name	***Type***	***Perm***	***Element***	***Units***
Now	Number	RO	JD	Julian Date
			UTC	UTC, hours
			UTCDate	UTC date*
			LT	local time, hours
			LST	sidereal, hours
			MoonAz	degs E of N
			MoonAlt	degs up
			MoonElong	degs E of sun
			MoonPA	paral ang, °+W
			SunAz	degs E of N
			SunAlt	degs up
Broadcast current time and sun/moon info at 2 Hz. Idle=unknown time OK=accurate Busy=acquiring Alert=time may be incorrect * packed as (year*10000) + (month*100) + (day)				
Location	Number	RO	Latitude	degs +N
			Longitude	degs +E
			Elevation	m
			MagDecl	degs mag-true
Observatory location. Sent once per connection. Idle=unknown OK=accurate Busy=acquiring Alert=location may be incorrect				
Site	Text	RO	Name	site name
Observatory site name. Idle=unknown site OK=accurate Busy=acquiring Alert=name may be incorrect				
Events	Number	RO	MoonRise[1]	JD*
			MoonSet[2]	JD*
			Dawn[1]	JD*
			SunRise[1]	JD*
			SunSet[2]	JD*
			Dusk[2]	JD*
Sun and moon rise/set information. Sent once per connection and when any value changes. Idle=unknown time OK=accurate Busy=acquiring Alert=info may be incorrect * -1 = never up -2 = circumpolar [1]previous rise/dawn if up now else next; [2]next set/dusk if up now else previous				

3.5 CCDCam

CCDCam – camera control				
Name	***Type***	***Perm***	***Element***	***Units***
MaxValues	Number	RO	ExpTime	seconds
			ROIW	raw pixels
			ROIH	raw pixels
			OSW	overscan pixels
			OSH	overscan pixels
			BinW	horizontal binning
			BinH	vertical binning
			Shutter	whether present
			MinTemp	Min cooler, C
Report maximum values for each camera operating parameter. Idle=unknown OK=valid Busy=looking up Alert=fault				
ExpValues	Number	WO	ExpTime	seconds
			ROIW	raw pixels
			ROIH	raw pixels
			OSW	overscan pixels
			OSH	overscan pixels
			BinW	horizontal binning
			BinH	vertical binning
			ROIX	raw pixels
			ROIY	raw pixels
			Shutter	whether to open
			Type	IMTYPE *
Set parameters for subsequent exposures. ROIW/H set to 0 implies full frame. Idle=unknown OK=valid Busy=checking Alert=invalid parameters for this camera * How IMTYPE FITS field will be set: 1=Bias 2=Dark 3=Flat 4=Science else not set				
Mode	Switch	RW	HiSpeed	1OfMany
			LoNoise	
Select hi speed or low noise mode for subsequent exposures. Idle=unknown OK=valid Busy=checking Alert=invalid parameter for this camera				
Shutter	Switch	WO	Open	AtMostOne
Directly control shutter. May not work on some cameras when exposure is in progress. Idle=no action OK=in position Busy=in progress Alert=error				
ExpGo	Switch	RW	Go	AtMostOne

CCDCam – camera control				
Name	***Type***	***Perm***	***Element***	***Units***
Send with Go=On to start an exposure as per last set of ExpValues. Send with Go=Off to abort. Idle=no activity OK=complete Busy=in progress Alert=error				
Pixels	BLOB	RO	Img	FITS file
FITS file sent when ExpGo completes. Idle=no file OK=file ok Busy=working Alert=error				

3.6 CCDChiller

CCDChiller				
Name	***Type***	***Perm***	***Element***	***Units***
Running	Switch	WO	Run	AtMostOne
Set whether cooler is running or in standby mode. Idle=unknown OK=command accepted Busy=setting mode Alert=error				
Remote	Switch	WO	On	AtMostOne
Set whether cooler is in remote or local (front panel) mode. Idle=unknown OK=command accepted Busy=setting mode Alert=error				
SetTemp	Number	WO	Target	°C
Set target cooler set-point temperature. Idle=unknown OK=target accepted Busy=checking target Alert=error				
CurrentTarget	Number	RO	Target	°C
Report current target set-point temperature. Idle=unknown OK=valid Busy=checking target Alert=error				
TempNow	Number	RO	Temp	°C
Report current cooler temperature. Idle=unknown OK=cooler at target temperature Busy=seeking target temperature Alert=error				
Status	Light	RO	TankLow	Tank level low
			FanFail	Fan failed
			PumpFail	Pump failed
			RTDOpen	RTD open
			RTDShort	RTD shorted
Report cooler status flags. Property status matches highest level of all constituents.				

3.7 1-Wire

Various devices on 1-wire bus via HA7Net				
Name	***Type***	***Perm***	***Element***	***Units***
Now	Number	RO	Humidity1	%
			DewPoint1	°C
			Temp1	°C
			Humidity2	%
			DewPoint2	°C
			Temp2	°C
			Humidity3	%
			DewPoint3	°C
			Temp3	°C
			Temp4	°C
			Temp5	°C
			RoofOpen	*
			RamOpen	*
Broadcast current stats whenever they change. * -1 = midway 0 = closed 1 = open Idle=not available OK=valid Busy=updating Alert=error				
JDAsk	Number	WO	JD	@ JD or 0=now
			ID	unique ID
Ask for historical Now stats nearest the given JD. Causes one AtJD response with matching ID. Idle=void OK=valid query Busy=looking Alert=invalid query				
AtJD	Number	RO	Humidity1	%
			DewPoint1	°C
			Temp1	°C
			Humidity2	%
			DewPoint2	°C
			Temp2	°C
			Humidity3	%
			DewPoint3	°C
			Temp3	°C
			Temp4	°C
			Temp5	°C
			RoofOpen	*

Various devices on 1-wire bus via HA7Net				
Name	***Type***	***Perm***	***Element***	***Units***
			RamOpen	*
			JD	JD
			ID	matching ID
Historical Now data at the given JD. Caused by JDAsk with matching ID. * -1 = midway 0 = closed 1 = open Idle=void OK=valid query Busy=looking Alert=invalid query				
Roof	Switch	WO	Open	AtMostOne
Open or Close Roof Idle=unknown state OK=accurate Busy=command in progress Alert=error				
Ram	Switch	WO	Open	AtMostOne
Open or Close Ram Idle=unknown state OK=accurate Busy=command in progress Alert=error				
Heaters	Switch	WO	On	AtMostOne
Turn Heaters On or Off Idle=unknown state OK=accurate Busy=command in progress Alert=error				
Fans	Switch	WO	On	AtMostOne
Turn Fans On or Off Idle=unknown state OK=accurate Busy=command in progress Alert=error				
Blind	Switch	WO	Open	AtMostOne
Open or Close lens blind Idle=unknown state OK=accurate Busy=command in progress Alert=error				

3.8 UPS

UPS – current state of Uninterruptable Power Supply				
Name	***Type***	***Perm***	***Element***	***Units***
Status	Number	RO	Battery	Percent charged
			MainsOK	1 or 0
Current state of UPS. Idle=unknown site OK=accurate Busy=updating Alert=error				

3.9 AC

AC – current state of Air Conditioning Unit				
Name	***Type***	***Perm***	***Element***	***Units***
Current	Number	RO	Temp	Current temp
			Set	Set point
Current status. Idle=unknown site OK=accurate Busy=acquiring Alert=error				
Set	Number	WO	Set	Set point
Set a new target temperature. Idle=unknown site OK=accepted Busy=in progress Alert=error				

3.10 QEx

Qex – Queued Execution				
Name	***Type***	***Perm***	***Element***	***Units***
Submit	BLOB	WO	Request	xml
Submit one new request. XML is packaged as a BLOB. Idle=unknown OK=command accepted Busy=checking Alert=error				
Schedules	BLOB	RO	All	xml
Report a complete set of all requests in database. XML is packaged as a BLOB. Idle=unknown OK=valid Busy=in progress Alert=error				
Delete	Number	WO	ID	Request ID
Delete the request with the given ID, Cancel is currently running. Idle=unknown OK=valid Busy=deleting Alert=error				
Cancel	Number	WO	ID	Request ID
Cancel the request with the given ID. Idle=unknown OK=cancelled Busy=canceling Alert=error				
Run	Switch	RW	On	AtMost1
Turn QEx schedule execution system on or off. Idle=unknown OK=running or idle Busy=changing state Alert=error				

3.11 Driver Intercommunication

Drivers communicate among themselves to perform coordinated operations, as show in Illustration 13. The FLI camera driver listens for meteorological data from the MAWS driver, building conditions from the 1-Wire driver and telescope pointing data from the Tel driver in order to add these values to the FITS image header. The Tel driver listens for meteorological data from the MAWS driver to compute the refraction model. The 1-Wire driver listens to the MAWS driver for Weather Alerts and the UPS driver for power failures to automatically close the roof and ram.

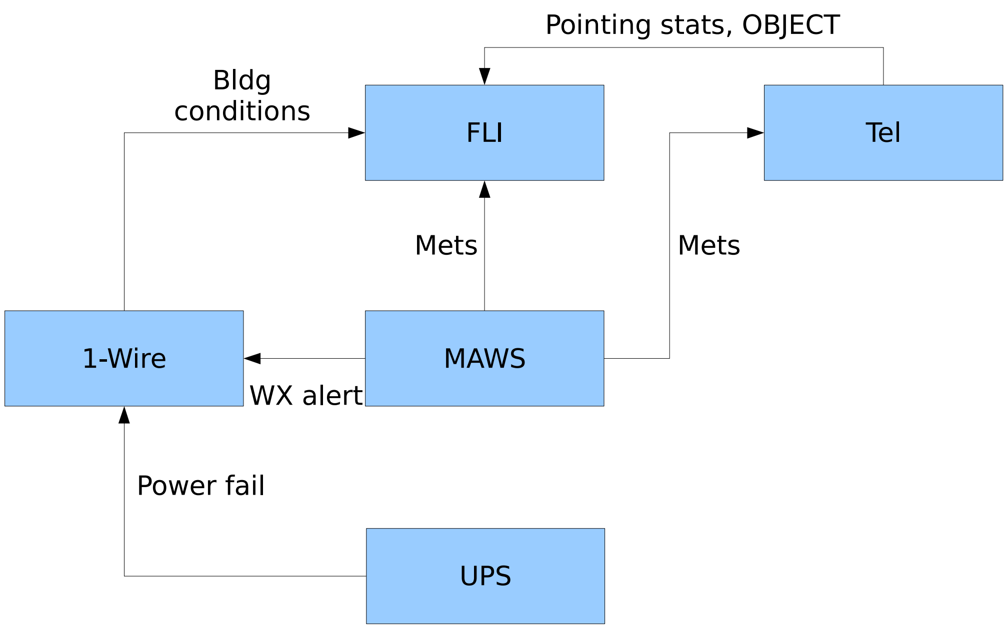

Illustration 13: Inter-driver communication

In addition to these connections, the qexcon driver also functions as a client. By this means it is able to issue all the same commands as any other client and thus operate all equipment at the observatory.

4 Command Line Programs

This section lists the major command line programs. Each command will provide a summary of itself when run with the –help command line option.

4.1 getINDI

getINDI connects to an indiserver and reports the current value of one or more properties. Values can be printed with or without corresponding names. GetINDI can also be used to monitor for changes in property values for an extended period. See the man page for full details.

4.2 setINDI

setINDI connects to an indiserver and sends commands to set new values for specified properties. When a property is an array that contains multiple elements, all elements are updated atomically. See the man page for full details.

4.3 evalINDI

evalINDI connects to an indiserver and listens for the the values of properties used as operands in an arbitrary mathematical expression then uses these values to evaluate the expression. See the man page for full details. The arithmetic expression follows the general syntax used in the C programming language. The operators supported include:

```
! + - * / && || > >= == != < <=
```

and the mathematical functions supported include:

```
sin(rad) cos(rad) tan(rad) asin(x) acos(x) atan(x) atan2(y,x)
abs(x) degrad(deg) raddeg(rad) floor(x) log(x) log10(x) exp(x)
sqrt(x) pow(x,exp)
```

4.4 pc48

This program allows direct control of the PC48. When run with no arguments it serves as a simple bridge, sending all characters read from stdin to the PC48 and sending all characters read from the PC48 to stdout. In addition, the program will ignore input lines that begin with the '#' character. This is to allow adding comments within a script file of commands. As a special case, an input line consisting of a single '!' character will display the control registers of the PC48.

The program also supports one command line switch, -r, which causes an initial reset of the PC48 before allowing normal commands.

It is possible to log to a file all commands going to and from the PC48, not only from this program but all programs on the system such as the inditel telescope control process. Whether or not logging is performed is controlled by the file /tmp/pc48lock. Logging is enabled if this file exists, contains at least one character and the first character is '1'. All other conditions, including not existing at all or other contents, result in no logging.

Turning logging on and off can be accomplished from a command line using the following example commands:

echo 1 > /tmp/pc48lock	to turn on logging

echo 0 > /tmp/pc48lock   to turn off logging

The log file itself is $OBSHOME/logs/PC48/<ISO-DATE>.log. There is one line per transaction. Each line begins with the UTC date. Following that everything sent to the PC48 is preceded with the character '>'. Everything received from the PC48 is preceded by the character '<'.

4.5 ik220con and ik220load

This program connects to the Heidenhain encoder controller, performs an initial setup, then goes into an infinite loop displaying the current values of the encoders. Each line of the loop output contains the following fields:

Axis channel number. This project has assigned channel 0 to HA, and channel 1 to Dec.

number of times the encoder returned the exact same value

The UNIX time in seconds since Jan 1, 1970

The current encoder count value

The encoder status, where 0 indicates normal operation. See Heidenhain documentation for other status values.

In order to function correctly, the ik220 linux driver module must be loaded. This module is normally loaded automatically when the system is booted. It can also be loaded manually using the command ik220load.

Note that there can not be more than one process reading the encoders at one time. This means, for example, you may not use this program while the inditel driver process is running.

5 Software Configuration

This section describes how the software is arranged both in terms of static disk files and in terms of dynamic operation, shown in Illustration 14. Relative path names are with respect to the OBSHOME environment variable which is /usr/local/octavi by default.

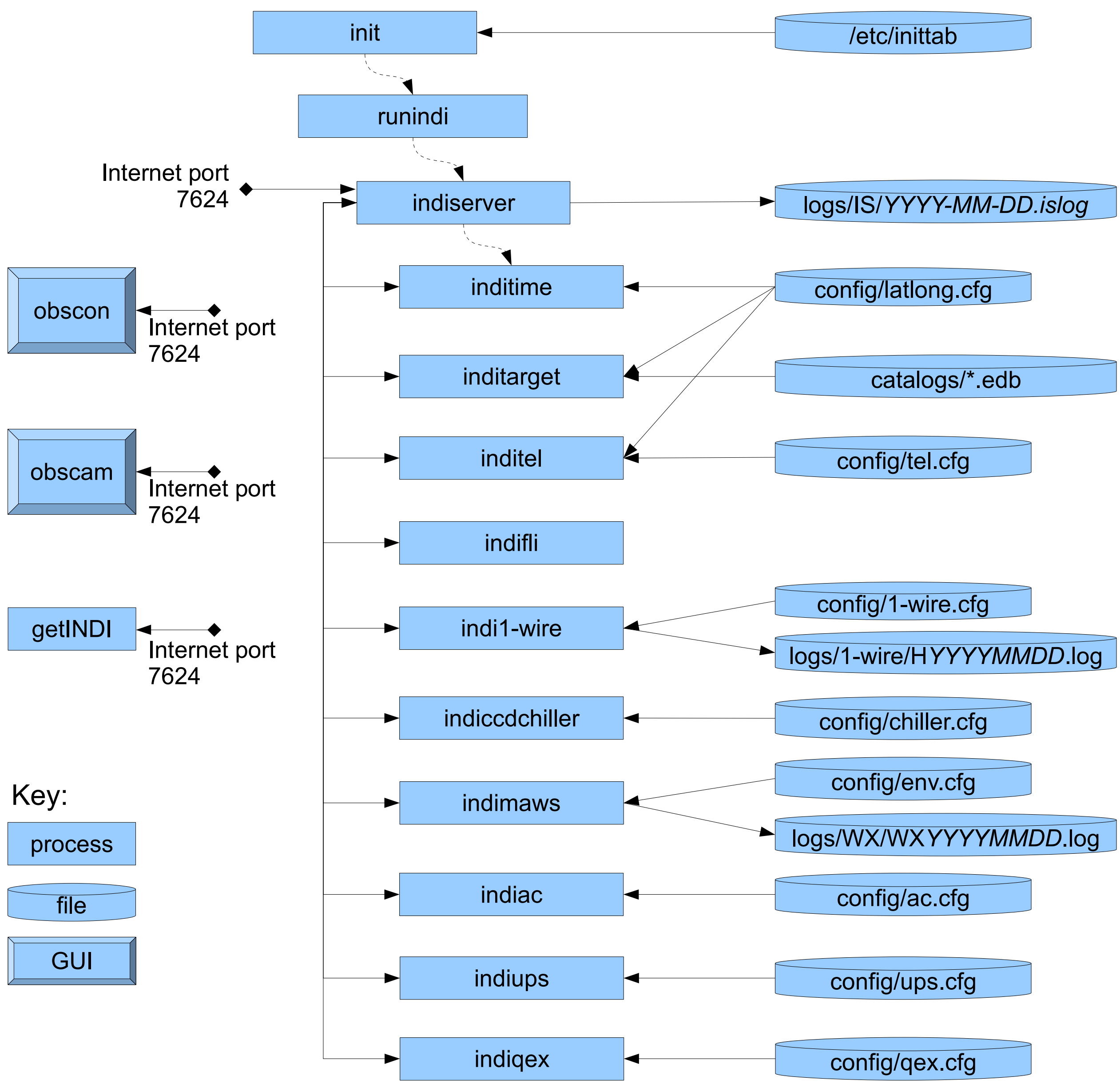

Illustration 14: System processes and files

5.1 Boot sequence

When the system is booted, runindi is executed at runlevel 5 via init from an entry in /etc/inittab. This starts the indiserver which starts each INDI driver. Indiserver and all drivers are expected to run forever. If any driver ever exits for any reason, it will be restarted immediately by indiserver. If indiserver itself ever exits, all its drivers will exit and indiserver will be restarted via runindi via init. The entry in /etc/inittab is as follows:

```
indi:5:respawn:su -s /bin/sh indi /usr/local/octavi/bin/runindi
```

If it ever becomes necessary to kill indiserver and keep it off, it will not work to just kill indiserver because init will start it again immediately via runindi. The correct way is to create a temporary file named /tmp/noindi and then kill indiserver as follows:

```
$ touch /tmp/noindi
$ sudo killall indiserver
```

The runindi script will see this file and just sleep for several seconds and exit. To allow indiserver to run again, just rm /tmp/noindi and runindi will then go ahead and start indiserver as usual. See runindi for details.

5.2 File system layout

The environment variable OBSHOME defines the root directory for a tree of all system files. By default it is set to /usr/local/octavi. This tree is organized into the subdirectories described in Table 2.

/usr/local/octavi	*Subdirectory Contents*
auxil	Supporting files
bin	Executables and scripts
catalogs	Catalogs
config	System configuration files
logs	System log files, including weather data
man	Man pages for indi related commands

Table 2: /usr/local/octavi contents

Auxil includes such files as models of natural satellites and geomagnetic declination.

Bin includes all indi programs and supporting tools, including drivers, obscon, the command line tools and the runindi boot script.

Catalogs include basic NGC, IC, Sky2000.

Config contains configuration files used by the INDI drivers. If it is desired to change any parameters, edit the file then kill the driver(s) that read it which will cause them to be automatically restarted and reread the new configuration.

Man contains UNIX style manual pages for the INDI scripting commands. To be accessible from the shell the path should be added to our MANPATH environment variable. For example, using csh syntax: setenv MANPATH "${MANPATH}:${OBSHOME}/man".

Logs contains subdirectories IS, PC48, WX and 1-wire.

- IS contains a trace record of all messages sent to INDI clients and diagnostic information from drivers. A fresh log is begun each day. The name of each file is of the form *YYYY-MM-DD*.islog. Each entry in the log begins with a time stamp in UTC.
- PC48 contains all traffic to and from the motion controller. Whether or not logging is performed is determined by the file /tmp/pc48lock. See §4.4 for more information.
- WX contains all weather statistics, both inside and outside the dome. A fresh log is begun each day. The name of each file is of the form WX*YYYYMMDD*.log. The format of the WX statistics files is fixed-width columns as defined in Table 3; all times are UTC.
- 1-wire contains the statistics from the four temperature and humidity sensors and roof and ram open or close status. A fresh log is begun each day. The name of each file is of the form 1W*YYYYMMDD*.log. The format is fixed-width columns as defined in Table 4; all times are UTC.

Column	***Field description***
1	Year
2	Month
3	Day
4	Hour
5	Minute
6	Second
7	JD
8	unixtime
9	Air temperature, C
10	Humidity, %
11	Dew point, C
12	Wind chill, C
13	Air pressure, hPa
14	Rain detected, 0 or 1
15	Rain accumulation, YTD, mm
16	Wind speed, m/s
17	Wind direction, degrees E of N
18	Recent wind max, m/s

Table 3: WX Weather log file format

Column	***Field description***
1	Year
2	Month
3	Day
4	Hour
5	Minute
6	Second
7	JD
8	unixtime
9	Humidity 1, %
10	Dew Point 1, C
11	Temperature 1, C
12	Humidity 2, %
13	Dew Point 2, C
14	Temperature 2, C
15	Temperature 3, C
16	Temperature 4, C
17	Roof status: -1 = midway, 0 = closed, 1= open
18	Ram status (same codes as Roof)

Table 4: 1-wire log file format

5.3 Building from Source Code

Login as user indi. This will set the OBSHOME, CVS_RSH and CVSROOT environment variables properly. OBSHOME is the global system directory for the executables and supporting files. It is normally set to /usr/local/octavi. The CVS variables are used to access the master repository maintained by Clear Sky Institute, Inc. Accessing this repository requires an account on the CSI servers.

The master source tree is in ~/octavi. If this directory does not already exist, download a new copy using the following command (access to the CSI servers will be required):

% cvs co octavi

Once the source tree is installed, subsequent updates are managed by the script ~/octavi/bin/buildall. This script has the following optional arguments:

- -u freshen the local copy from the CSI repository
- -c remove all local temporary and derived files by invoking `make clobber`
- -b build all programs in the local source tree by invoking `make pass1-6` in order.
- -i install the executables in the global OBSHOME tree by invoking `make install`.

For example, to update the source tree and build and install everything use the following command:

```
% ~/octavi/bin/buildall -u -b -i
```

This works in conjunction with standardized Makefile targets. Most directories in the source tree contains a Makefile. Each Makefile may contain one or more of the following targets:

clobber
: This target removes all temporary and all derived files from this directory, leaving only files that constitute original material.

pass1

pass2

pass3

pass4

pass5

pass6
: These six targets are used to perform sequential operations during the build process. Pass1 performs any necessary prebuild steps. Pass2 builds the documentation. Pass3 makes all libraries. Pass4 makes daemon processes and device drivers. Pass5 makes command line programs. Pass6 makes GUI programs.

install
: This target installs everything in the global OBSHOME tree.

6 Hardware Connections

This section describes how the software assumes the hardware is connected. The overall topology is shown in Illustration 15.

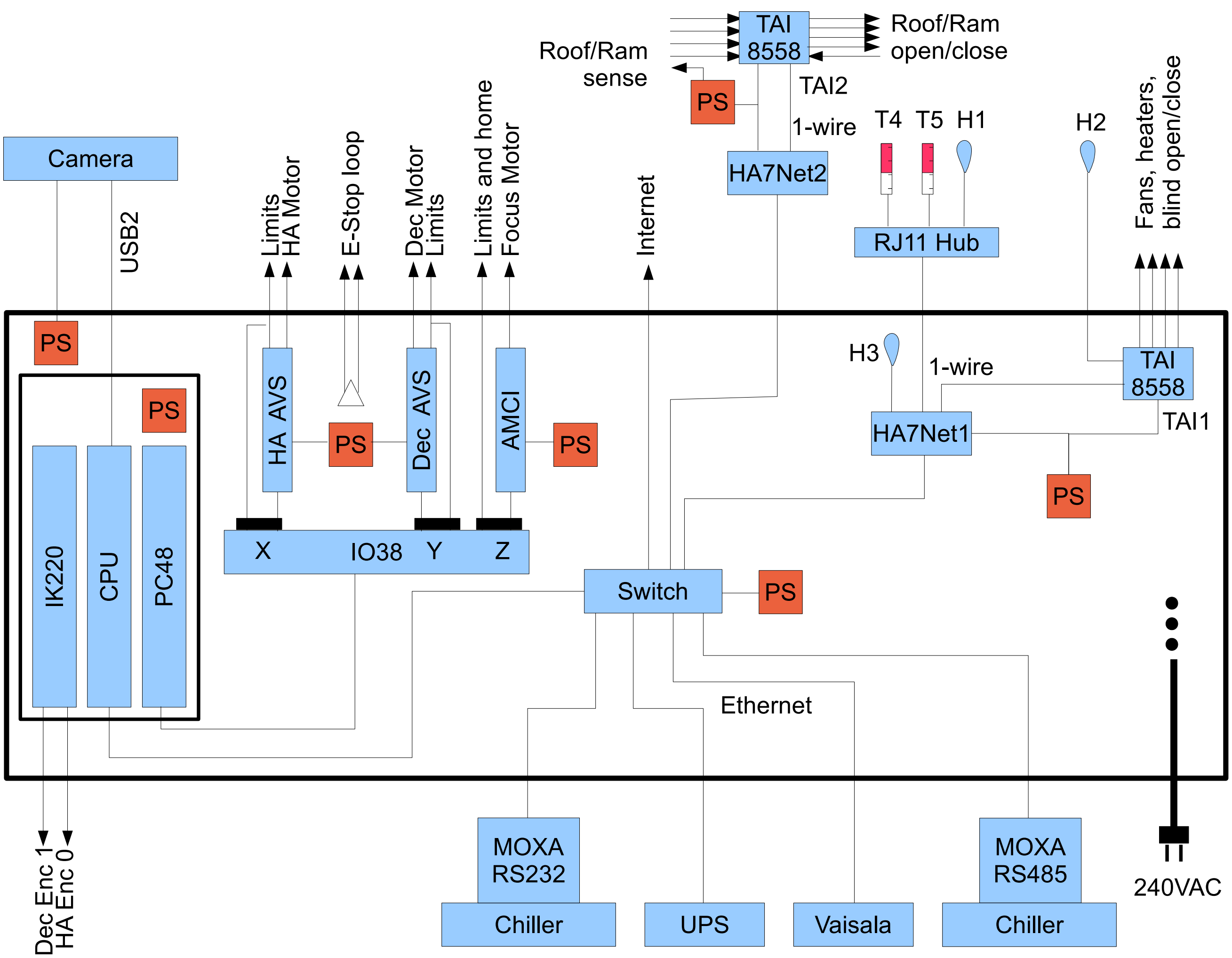

Illustration 15: Overall electrical diagram

All equipment within the bold outline should be located within a shielded aluminum case. The case should be well connected to building ground. Devices with overlapping voltage ranges should share power supplies where ever possible to save space. All ethernet and 1-wire cables that penetrate the case should go through RJ45 bulkhead connectors; note that the 4-pin 1-wire modular connectors can use the same pass through part as the 8-wire ethernet connectors. Similarly all other cabling should go through circular twist-lock or threaded connectors. See following sections for specific pin assignments and further details as appropriate.

6.1 PC48 Motion Controller

The PC48 is an ISA motion controller board installed in the main computer chassis. It must be set to bus address 360, with the lower 8 user bits jumpered as input, the upper 8 as output.

The HA motor is connected to the X axis via an AVS servo amplifier, with positive and negative limit switches that connect to ground when active on both the AVS and the IO38 DB9. The number of steps per revolution is defined by the parameter mxspr in the file tel.cfg, with sign positive if counts increase with HA.

The Dec motor is connected to the Y axis via an AVS servo amplifier, with positive and negative limit switches that connect to ground when active on both the AVS and IO38 DB9. The number of steps per revolution is defined by the parameter myspr in the file tel.cfg, with sign positive if counts increase with Dec.

The HA and Dec control loops must be properly tuned for proper performance. Each control loop commands its axis to a velocity equal to the velocity of the target plus an amount proportional to the following error. The constant of proportionality is Kxp and Kyp, respectively. The maximum velocity and acceleration allowed for each axis is specified by Vxmax, Axmax, Vymax and Aymax. The Focus motor is connected to the Z axis, with positive and negative limit switches that connect to ground when active. One of the limit switches is also wired to be the home switch. In the tel.cfg configuration file, parameter fhnlim sets which limit is acting as home. The first time a focus position is commanded the focus motor will first be homed to the specified limit position then the position will be located. From then on homing is not performed unless the inditel driver is restarted. The parameter fspum sets the number of steps/µm of travel; fvmax and famax set the maximum velocity, in steps$\cdot$s^{-1}, and acceleration, in steps$\cdot$s^{-2}, respectively. The acceleration should be sufficiently large that the motor does not coast too far into the limit switches when activated.

The focus motor is connected to the Z axis via an AMCI stepper controller. The positive and negative limit lines go low when active and are connected to the PC48 DB9. The home line is connected in parallel with the negative limit switch.

6.2 Emergency Stop

An observatory emergency stop line can be sensed by the control system on the input line defined by EStopIBit in tel.cfg, which is bit 0 by default. While E-Stop is present, the control system will command both telescope axes to stop and will indicate E-Stop is active on the GUI display of ObsCon. But in addition it is expected that the E-Stop line is also wired *directly* to the AVS power supply to stop the mount drive motors. It may also be wired to the roof and ram power supply to stop their motions if desired.

Illustration 16 shows a concept circuit. The idea is to provide a series loop circuit that must remain closed in order for power to reach the AVS servos controlling the telescope motors. If this circuit is open for any reason power is removed and the telescope must stop. Using a closed loop provides a degree of fail-safe operation because a fault anywhere in the loop, such as a broken wire, bad connector or stuck switch, activates the stop action and is immediately apparent.

All switches in the loop are normally closed and the normally open relay contacts are used. Four of the switches in the loop are strategically placed on the telescope mount so they open when the telescope is at an extreme position and must not move any further. Two of the switches are of the red mushroom style used as industrial emergency stop switches, mounted on the wall of the telescope and control rooms. A third relay is placed in the loop that connects to the PC48 I/O pin 0 to allow the control system to know when the emergency stop has been activated. Relays to cut power to the roof and ram are not shown but could be added easily if desired. Relay power is not specified but is expected to be whatever is suitable for the relay coils chosen for the project.

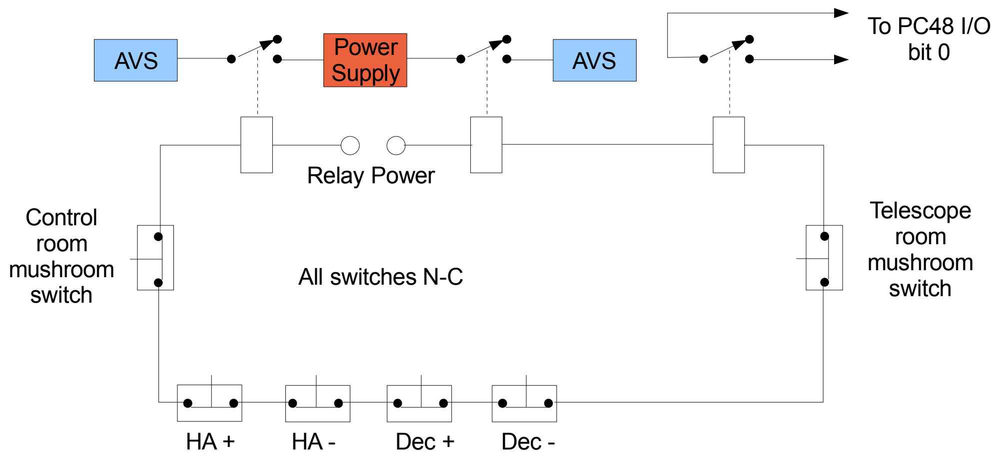

Illustration 16: E-Stop concept circuit

6.3 Roof and Ram Control

The Roof and Ram controller is connected to a TAI8558 1-wire relay and sense module. There is one relay for each of four functions: Roof open, Roof close, Ram open and Ram close. There is one input line to sense each of these functions as well. This TAI8558 is on its own HA7Net in order to reduce 1-wire cable lengths. Network configuration parameters are defined in the 1-wire.cfg configuration file. The relay and sense connections are assigned according to Tables 5 and 6, respectively. Note that each input opto isolated line requires a pullup resistor to a source of voltage, simple circuit closure is not enough. The same power supply used for the TAI may be used for this purpose also. The software assumes that the sensors are shorted (no voltage) when active.

Relay output Jack	***Purpose***
J6 2-3	Roof open
J7 2-3	Roof close
J8 2-3	Ram end open
J9 2-3	Ram end close

Table 5: Roof control relay assignments

Opto isolated input Jack	***Purpose***
J2	Roof is open
J3	Roof is closed
J4	Ram end is open
J5	Ram end is closed

Table 6: Roof sense input assignments

Due to the inability of the pneumatic system to reliably operate the roof and ram at the same time, the roof is given priority over the ram. Thus ram operations in progress are suspended during roof operations and new ram operations are held pending if roof operations are in progress.

6.4 OTA Equipment

The optical tube assembly contains fans, heaters and a cover blind connected to a TAI8558 relay module on the 1-wire bus. Network configuration parameters are defined in the configuration file 1-wire.cfg. Table 7 shows the assignments of the TAI relays to each function.

Relay output Jack	***Purpose***
J6 2-3	Fans
J7 2-3	Heaters
J8 2-3	Blind open
J9 2-3	Blind close

Table 7: OTA Fans, heaters and blind assignments

The TAI5885 is located inside the main case, and only the switched power lines run to the equipment.

6.5 IK220 Encoder Input

The two Heidenhain absolute encoders on the mount are connected to an IK220 PCI card installed in the computer chassis. Since they are absolute encoders, no homing sequence is required. Each encoder provides 25 bits of angular precision, or 25.89 counts per arc seconds.

The HA encoder is connected to the axis specified by the parameter named exaxis in tel.cfg. The Dec encoder is connected to the axis specified by eyaxis. These are 0 and 1 by default, respectively. The parameters exspr and eyspr specify the number of steps per complete revolution of the axes, respectively. The signs are positive for increasing HA and Dec. The reference positions are captured in the mount model.

6.6 Camera Chiller

The camera can be cooled by an auxiliary Lytron Thermocube 200 chiller. Its serial control line is connected to the control computer via a MOXA RS232-ethernet converter. The MOXA is mounted inside the chiller and derives power from the chiller. The network address for the MOXA is defined in the chiller.cfg configuration file. The file also contains the parameters ontarget for setting the temperature difference considered to be on target and check for setting how often to poll the chiller for status information.

6.7 UPS

The UPS is connected to the system via ethernet. The IP address is defined in the ups.cfg configuration file. The file also contains the parameter `holdtime` which defines how long the UPS is allowed to be on battery before shutting down the telescope.

6.8 Air Conditioner

A Ciatesa air conditioner is connected to a serial port via a MOXA RS485-ethernet converter. It is expected that the MOXA is mounted inside the air conditioner chassis and derives power from the air conditioner circuitry. The IP address is defined in the ac.cfg configuration file.

6.9 Temperature and Humidity Sensors

Two temperature and two humidity sensors are connected via one HA7Net[7] using 1-wire[8] buses. The HA7Net forms a bridge between the 1-wire bus and the observatory ethernet LAN. Any sensor based on the HIH-4000 humidity sensor, manufactured by Honeywell, and using the DS2438 IC from Dallas Semiconductors, will work. Network configuration parameters are defined in the 1-wire.cfg configuration file.

6.10 Camera

The system supports one CCD camera made by Finger Lakes Instrumentation[9] connected via any available USB port. If camera performance is less than expected, try using a different USB port. The power supply for the camera is located inside the main equipment box.

6.11 Cabling and Grounding

All cabling from the main equipment box destined for the telescope should be fastened to the main telescope support beam and routed towards the south end of the hour angle gimbal bearing where the HA motor is located. Cables that then continue on should be fastened to the gimbal and routed towards the east declination bearing where the Dec motor is located. Cables that then continue on to the optical tube assembly should be routed to their final destination. All cabling should be fastened securely and neatly into bundles. Cables should be grouped into separate low voltage signaling and high power bundles. Cable bundles that pass by the HA or Dec bearing should be formed into a loop with the minimum length necessary that safely avoids any contact with moving parts.

All equipment should be connected to the building lightning ground bus bar which should be separate from the electrical supply ground. All ground connections should be made with heavy copper strap and connected using teethed lock washers. All ground straps should be as short and direct as possible. A separate ground strap should be installed around both the Dec and HA bearings so stray current is not required to flow through the bearing itself. All paint must be

7 http://embeddeddatasystems.com/page/EDS/PROD/HA/HA7Net
8 http://www.maxim-ic.com/products/1-wire
9 http://www.fli-cam.com

removed around ground mounting holes and the metal cleaned of all grease and debris before the bolt, lock washer and nut are tightened for the last time. Once tightened, the connection should never be loosened again to help maintain a gas tight corrosion resistant connection.

7 Document History

Version	*Date*	*Author*	*Changes*
1.0	2007-6-20	E.C.Downey	original draft
1.1	2007-7-20	E.C.Downey	Start overview. Add more devices. Start CL section.
1.2	2007-12-9	E.C.Downey	Add Camera.Shutter
1.3	2007-12-30	E.C.Downey	Add hardware and software sections
1.4	2008-1-25	E.C.Downey	Update Environment driver; add Telescope.SetVelocity
1.5	2008-2-1	E.C.Downey	Update diagrams.
1.6	2008-11-28	E.C.Downey	Minor edits before site visit.
1.7	2008-12-16	E.C.Downey	Update hardware description
1.8	2008-12-29	E.C.Downey	Time.Location.Longitude is +E. Roof/Ram wiring.
0.8	2009-5-29	E.C.Downey	Roll back this numbering; start section for qexcon.
0.8.1	2009-8-29	E.C.Downey	Put TAI8558 for roof on its own HA7Net
0.8.2	2009-9-13	E.C.Downey	Update qex scheduling algorithm
0.8.3	2009-10-8	E.C.Downey	Add CCDChiller.Remote property.
0.8.4	2009-11-28	E.C.Downey	Add 1-wire to driver diagram. Add Tel.Pointing.PA/X/YVel
0.8.5	2009-12-28	E.C.Downey	Add build instructions. Add CCDCam.ExpValues.Type
0.8.6	2010-02-14	E.C.Downey	Add 1-wire H3
0.8.7	2010-08-29	E.C.Downey	Add SatIsSunLit QEX constraint
0.8.8	2010-09-06	E.C.Downey	Describe QEx canceling; change PropPix to PropSave.
0.8.9	2010-09-12	E.C.Downey	Define /tmp/noindi
0.9.0	2010-09-20	E.C.Downey	Describe roof has priority over ram control.