LBT Telescope Control System
GUI Guidelines

M.D. De La Peña

Second Version: 14 August 2005
LBT CAN 481s010b - DRAFT

1 Introduction

In order to gain some perspective and understand the choices made in terms of the design for the telescope control system (TCS) graphical user interfaces (GUIs) for the Large Binocular Telescope (LBT), it is useful to have background information on the overall software and hardware configuration. This document describes at a high level the general TCS software and hardware design. References to other documents which provide detailed discussions of the software and hardware issues are given herein.

2 Software Architecture

The LBT system is segregated into major software components which primarily correspond to hardware components (e.g., enclosure, primary mirror cell, guiding); the software components are hereafter referred to as the LBT "subsystems" (Axelrod 2002). Each subsystem is written in C++ and executes on a Linux platform which it may share with other subsystems. Each LBT subsystem communicates with its corresponding software/hardware (e.g., PLC, VxWorks) via methods appropriate to those systems.

There is a dedicated GUI for each subsystem which runs as an independent program. The term GUI is used to describe the collection of main windows, dialogs, and panels which comprise the user interface to a subsystem. Although the GUIs are independent executables, overall they have been developed in an integrated fashion with their respective subsystems. The GUIs and their underlying support software are created using Qt Designer (Trolltech). Some general characteristics of the GUIs are:

- GUIs for individual subsystems do *not* communicate with one another.
- A GUI *never* communicates directly with its own subsystem or to any hardware.
- The GUIs issue commands as XML strings which are transferred to a command sequencer (CSQ) using a *customized* remote procedure call (RPC) library. The customized library properly supports a multi-threaded environment and only needs to accommodate the transfer of strings.

- When a command is sent to the CSQ, the GUI receives a unique handle from the CSQ which allows it to poll and determine the status of the command.

- Each subsystem GUI typically obtains its state and other critical information from network shared memory.

When a GUI program is invoked, the GUI is instantiated in its "design-time" form and in a *disabled* state. This is done as a safety measure since the state of the telescope is not known a priori. Once the GUI software is able to determine successfully the status/state of the telescope from network shared memory, the GUI updates itself with the current information and becomes enabled. It should be noted that any TCS GUI can be aborted and restarted without impact on the TCS or the telescope.

Part of a future implementation scenario includes the use of "subsystem checklists". In the event of operational problems, each subsystem GUI generates a "subsystem checklist" pertinent to the problem at hand to aid the operator in resolving the issue. The checklist consists of clear and concise steps the operator must follow to resolve the problem. It is also designed to force the operator to perform certain steps in a particular order (if necessary). The checklist concept is based upon the checklists generated by modern-day aircraft control software.

3 Hardware Architecture

All TCS software is installed on all mountain servers, as well as the main telescope console. However, the servers are reserved for running the subsystems, and the main telescope console is reserved for execution of the GUIs.

A goal for the LBT TCS GUIs is to be loosely-coupled, yet strongly cohesive with their corresponding subsystems. They are loosely-coupled in that they are separate executables from their subsystems and are an extreme form of the "model/view/controller" paradigm; the GUIs could be replaced with different views created by any number of GUI toolkits. However, the GUIs are also strongly cohesive as they are being developed (for the most part) as an integrated part of the subsystem software (e.g., the subsystem is not being developed and then a GUI added later as an afterthought).

The advantage of having a specified "GUI platform" are:

- the ability to utilize more capable graphics cards than the standard set of subsystem machines

- rapid interactive response (most communication done by asynchronous request and no delays due to a busy machine)

- segregation of remote user access (via VNC) to the GUI client rather than direct access to the subsystem machine (possibly better security and no use of subsystem resources)

- VNC server can be implemented and run only on the GUI machines

- subsystem GUI always available for display but is disabled if the subsystem is not accessible, and
- GUI machines can be rebooted as necessary without disruption to the TCS subsystems.

In essence by putting the GUIs on their own platform, the GUIs and remote access services would be segregated from the workhorse subsystem machines.

3.1 Main Telescope Console and TCS Displays

The main telescope console or telescope operator (TO) station consists of three, 24.1-inch, high-resolution (1920x1200), flat panel LCD displays being driven by a single Linux machine. The displays are configured to work in conjunction with one another as a single large display. In this way, overall system status, monitoring of telemetry, and other critical GUI status panels can be viewed at all times on one of the displays while the operator is actively working with panels and dialogs on the remaining two console displays. Section 5 provides some preliminary TCS GUIs. For other individuals and visitors present in the control room, three large (40-inch), medium-resolution flat panel LCD monitors mounted on the control room wall will be used to display general system status, weather, and astronomical information.

4 GUI Guidelines

This section defines the guidelines used in the creation of the LBT GUIs.

4.1 Font and Colors

The LBT GUIs are all running on Linux platforms where the Helvetica font is available. If this font is not available for use on your particular platform, please make sure to use a San Serif font such as Arial or Verdana.

- Use Courier font with a font style of bold for numerical values.
- Use Helvetica font with a font style of bold for text.
- Use a font size of 12 or 14 point for most items on the panels and dialogs associated with the subsystem "main window". Use a larger point size, as necessary, for heading labels and other major items.
- Most static labels are written in mixed case text, except for static labels annotating major items which are in uppercase text. Indicator text relating system state (e.g., OPEN, CLOSED) is in uppercase, though there are exceptions in order to better distinquish between possible values.
- Color should be used sparingly. The currently defined LBT colors are listed in Table 1. The *Red, Green,* and *Blue* columns contain values which range from 0 through 255.

- Red is reserved for the error/alarm situation. Red should *only* be present on a GUI in order to indicate an error or alarm.

Color Name	Red	Green	Blue	System Status	Purpose
dark gray	214	214	214	Off/unknown	default background
blue	0	0	255	On/Off	informational text
green	100	255	100	On	ok/normal/nominal
yellow	255	255	0	On/Off	warning
red	255	0	0	On/Off	error/alarm

Table 1: LBT defined colors.

Some GUIs employ two pixel maps representing the error/alarm alert and the emergency stop indicator. These pixel maps are illustrated in Figure 1. These pixel maps are available on request. Further, some GUIs use a very lightly textured, gray background which is also available on request; the weather (WX) and telemetry (TEL) GUIs both use this textured background.

Figure 1: Pixel Maps for the alarm indicator and panic or emergency stop.

4.2 Design-time Issues

- Since each GUI is an individual executable, each subsystem GUI should consist of a "main window" and ancillary support dialogs, as necessary. The main window contains the menu bar which should provide (at the least): a way to shutdown the executable gracefully (i.e., **Exit**), and access to assistance for the GUI (i.e., **Help**).
- Use widgets appropriate for the input data type. For example, string input can be done via an entry, integer values can be entered using a spin box, and a long list of specific choices can be represented by a combo box. Using widgets as appropriate to the input data type can have the advantage of automatic input validation and minimize the need for parsing or use of regular expressions. Figure 2 illustrates a simple example for integer input.
  - Although Qt Designer allows one to set the minimum and maximum limits for certain widgets, the limits should **NOT** be hard-coded into the GUI design.

Rather, the limits should be obtained from reflective memory and set in the GUI code.

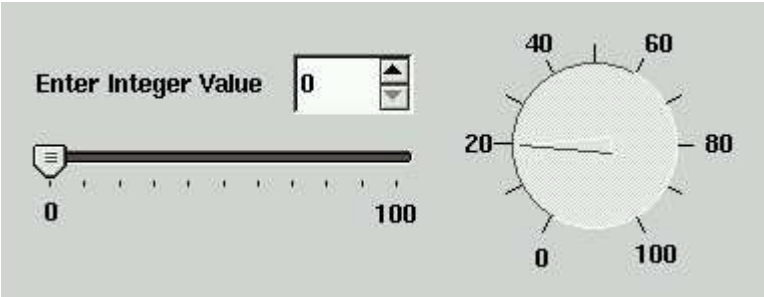

Figure 2: Example of widgets for entering integer values.

- Use action labels for all buttons (e.g., OPEN, STOP, DISPLAY). If appropriate, labels should contain more descriptive text (e.g., SET TEMPERATURE, SET OFFSET).

- Differentiate clearly between control and indicator fields. Control fields are to obtain information from the operator; indicator fields are to display read-only information from the system. In specific, string control fields are represented by an entry (aka line edit, text entry box) widget. These widgets typically are characterized by a sunken relief, and have a white background with black foreground text. Indicator fields should be represented so they *do not resemble control fields.* The LBT TCS indicator fields are actually label (aka static text) widgets with a raised edge relief.[1] The background and foreground colors depend upon the system status.

  ok/nominal - black background with green foreground text

  warning - yellow background with black foreground text

  error/alarm - red background with black foreground text or dark gray (default) background with red alert pixel map

  In practice, whether one uses yellow background with black foreground text or black background with yellow foreground text to indicate a warning is dependent upon the specific GUI. The main issue is to ensure that the information is highlighted in a manner easily interpreted by the user.

  Figure 3 illustrates an input entry and an indicator widget with the ok/nominal colors. The central item in Figure 5 (the alert symbol followed by the red ALARM text) is an example of the indicator widget for the error/alarm situation. It should be noted that the indicator fields for the global telescope state (as seen in Figure 9) are deliberately unique in appearance. These indicator fields are still label widgets with a raised edge

[1] There is a misconception that a label widget, a non-editable text component as rendered on the GUI, is only to be used to label other items on the GUI; this is not true. This misconception leads to the error of programmers using the entry widget to display read-only information.

relief. However, the foreground text is in black and the background is green, yellow, or red depending upon the situation.

Figure 3: On the left is a control (input) field; on the right is an indicator (read-only) field.

- Ideally, color alone should not be used to distinquish important items or state. Some of the TCS GUIs use **both** color and text/symbol as indicators. However, having two labels for every indicator uses a lot of GUI real estate, so many of the GUIs do not employ this technique. For most variables in the system, the ECS GUI uses two indicator widgets labeled STATUS and STATE. The STATE widget always contains *only* a color. The STATUS widget contains text and/or a pixmap depending upon the situation. Some examples are shown in Figures 4 and 5.

- A blinking alarm symbol or color can also be used to indicate extreme situations.

Figure 4: Status text and state color indicators for ok/nominal performance.

Figure 5: Status text and state color indicators for the error/alarm situation.

- Do not hard-code colors into the code. The colors should be defined as variables or objects which are set at a higher level. This is to allow for possible alternative color schemes which would accommodate color blindness or a refinement in the default system colors.

- Clearly label all items.

- Indicate units of all values. Regardless of the actual units used by the subsystem, all units represented on the LBT GUIs must be the International System of Units (SI) (metric).

  – Do **NOT** provide the option for the TO to change the GUI from displaying SI/metric to displaying British/Imperial units. With experience, the TO and other users become sensitive to the order of magnitude of a value rather than its units. Allowing the TO to change units only serves to confuse and forces the use of units when quoting the value.

- Group items, as appropriate, for clarity. Group boxes or other container widgets can be used for this purpose.

- Make sure all modeless dialogs have a CLOSE button, and all modal dialogs have OK and CANCEL buttons to shutdown the panel/dialog. The operator should **not** have to rely on the use of the "X" on the dialog title bar.

- Use variable names for the widgets which reflect the purpose of the widget. In particular, the suffix should indicate the nature of the widget. Variable name *vdOpenAllBut* represents the *ventilation door, open all button.* Variable name *dpLSStatusValLab* represents the *left-side damper status label* used to display the *status of Running or Stopped.*

- As noted in the **Run-time Issues** section, radio buttons and entry widgets are for selecting an option or typing input values. These widgets should **NOT** be attached to callbacks or slots which send the values into the system. These widgets must be associated with an action button (e.g., SET TEMPERATURE) which actually invokes the callback or slot. As the interface to a control system, we want to provide a means for the TO think about their choices before the choices are set into the system.

- You must create a passive or *read-only* mode for any GUI which can invoke actions. Although the complex levels of safety and security should most correctly be handled by low-level routines in the service layer of software, this model is currently not implemented. As an interim solution, a passive version of the GUI must be implemented for safety. The ECS GUI has a simple, passive model implemented which allows the user to navigate all ECS panels and dialogs without worry of accidental action invocation.

4.3 Run-time Issues

- Make sure when the subsystem GUI shuts down, all child dialogs also shut down.

- Use widgets according to their function. For example, radio buttons should be used only for choosing options; they should not trigger actions which are transmitted to the underlying system.

- An entry widget is for entering free-form text. The operator action of pressing the carriage return upon entry completion or moving the mouse out of the widget should *only* trigger possible entry validation and **not** an action which is transmitted to the underlying system.
- Do not use toggle buttons. In particular, do not use toggle buttons which change their text label when pressed (e.g., a single button which reads ON or OFF depending upon the state of the system).
- Do not disable action buttons based upon the most recently chosen action. For example, given OPEN and CLOSE buttons, the operator presses an OPEN button to open the shutter doors. Once the action is completed, do not disable the OPEN button. Although under most circumstances it would be incorrect for the operator to press OPEN again, it is the responsibility of the subsystem to check the state of the subsystem and respond appropriately (i.e., by ignoring the command).
- Only use ellipses ("...") when more information is needed before the specific request can be fulfilled (e.g., SAVE AS...). Do not use ellipses just to indicate that a dialog will be invoked.
- If an action is in progress, keep the user informed that something is happening. If there is an actual way to track the progress, the user can be kept up-to-date with a progress bar widget. For actions which cannot be tracked while in progress, a slow flashing of the STATE indicator field color (e.g., alternating between the default background color and green), or the text "Running..." could be used. The goal is to let the operator know the command was received, and the system is working toward fulfilling the request.
- Implement keyboard short cuts and accelerator keys.
- Strive to minimize mouse motion within a panel.
- Mouse-less navigation (TAB, SHIFT-TAB) is typically provided by default with most GUI toolkits. The programmer just needs to ensure the navigational sequence between each control widget is orderly. To make full use of mouse-less navigation, implement the "key press" action for buttons so the user does not have to depend upon the mouse.

4.4 At Least One More Thing...

In the past, it was required that the last step of the design process of the GUI was to render the *contents* of the main window and any panels/dialogs which are available when the interface is first invoked as *disabled*. This is no longer strictly necessary as every GUI should check the state of its corresponding subsystem upon instantiation, as well as during its regular update cycle. If the underlying subsystem for a GUI is not active, the GUI should disable all of its associated windows/dialogs/panels and render itself in the alarm color as defined in Table refT1. Disabling the GUI is for safety, and an entire GUI depicted

in the alarm color is truly an alarming (horrible) sight. This rendering has proven to be sufficient to get the attention of the TO.

4.5 Allowed Technologies

In an effort to keep a handle on the number of technologies used, and hence, supported for the GUIs, the following is the current short list of allowed packages:

- Qt Designer - GUI Builder (Do **not** use any 3D aspects at this time.)
- Matlab/GUIDE - Utility for rendering the graphics; GUIDE is a GUI Builder associated with Matlab that will need to be used for adding controls to the graphics.
- DS9 - Astronomical Image Display Server
- XEphem - Astronomical Software Ephemeris
- GIMP - Image manipulation packet (This was used to create a low-key, textured background used by some of the GUIs. This should not be used in general.)

It is understood as the development of subsystems and their corresponding GUIs progress, it may be necessary to employ additional technologies. The new technology and design **must be discussed before any implementation.**

5 TCS GUI Examples

This section contains a few preliminary GUIs for the LBT TCS, and is included to provide a fuller context for the LBT guidelines. These GUIs are for illustration purposes only. The GUIs are running on machines with incomplete simulators, so the information displayed on the GUIs is not necessarily consistent or correct. Figure 6 shows the Enclosure Control System (ECS) GUI. This particular subsystem consists of many mechanical components, and a tabbed widget was chosen as a way to present the controls in a compact fashion. In this example, the tabbed widget is open to the Shutter Doors page. This page contains action buttons for opening and closing the doors, and clearing alarms, as well as buttons which invoke ancillary dialogs; All the buttons are labeled with action terms (verbs). The STATUS widget contains either text or a pixel map (e.g., alarm), and the STATE widget contains an appropriate color. Since the position of the shutter doors can be tracked, a progress bar widget has been employed to show the progress of the issued command. The Shutter Door system provides many details associated with the running of the doors. Under normal circumstances, there is no reason to view these details. However, the DISPLAY DOOR DETAILS button allows the TO to reveal or hide the details from the main Shutter Door control page as necessary.

Since this GUI is being used as an example, one other feature of this GUI is worthy of note. The top of the GUI employs the use of "eyebrow" control. The "eyebrow" consists of a string of small action buttons which reflect the corresponding ECS subcomponent by

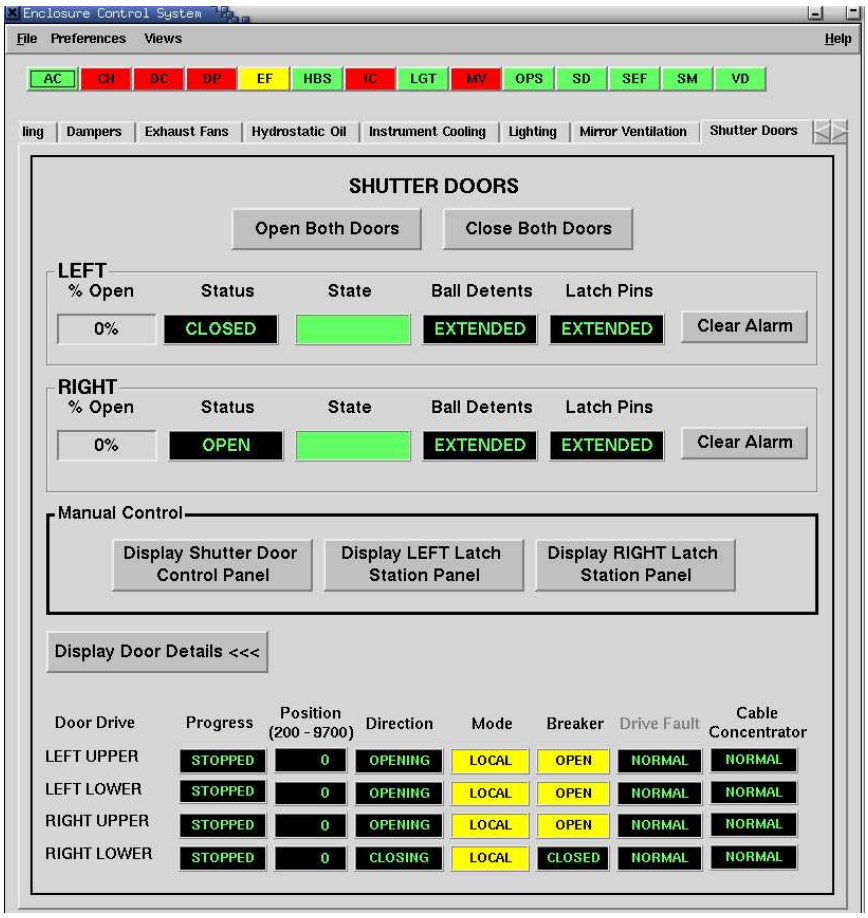

Figure 6: Enclosure Control System, Shutter Doors main window.

name, the state of the component by color, and the buttons provide fast navigation to the tabbed widget page.

Figure 7 is another page of the ECS tabbed widget showing Instrument Cooling. This figure shows the entire ECS GUI; note the message box at the bottom of the page. Although all alarms/warnings are to be displayed on a "Master" GUI which monitors state for the entire TCS, it has proven useful to display specific alarms on the main window of the particular subsystem GUI. From the "eyebrow" indicator, the TO can see there is a problem on the **IC** or Instrument Cooling page; the TO can quickly navigate to the page by pressing the "eyebrow" button. Not only are the Water Pumps in alarm as seen by the Status/State indicators and the information in the message box, but there are also problems with the valves. The Display Valve Control Panel is displayed in the alarm color as a "breadcrumb" to lead the TO to the ancillary panel which shows the problem for the mechanism.

This page can also be used to enter new Temperature Setpoint and Deadband values.

In order to transmit the new values to the underlying system, the operator must press the appropriate SET button. The new value will be displayed in the indicator field once the request has been accomplished and the value is read back by the GUI update routine.

Figure 8 is an example of the help system invoked from the HELP menu of the PMC GUI. The help is rendered using the QAssistantClient class; this class provides a mechanism to use the Qt Assistant as part of the GUI application tool.

Figure 9 shows a prototype for the Main Telescope Console main window. While this window consists mainly of many indicator fields, the critical widgets resident on the window are the controls for setting the global telescope state and for invoking the emergency stop. As noted in Section 4.2, the indicator widgets for the global telescope state are unique with respect to all other indicator widgets. The emergency or panic stop employs the use of a yellow and black crosshatch pixel map specifically designed for the purpose. Finally, the main console has a system messages area for the display of major system activity. This message area will contain a filtering control so messages of various priorities (or degrees of emergency) can be displayed.

6 References

1. Axelrod, Tim 2002, "LBT Software System Definition", LBT CAN 481s001.
2. Kraus, Joe 2003, "TCS Network and Computer Hardware Plan", LBT CAN 480s020a.
3. Qt Designer, www.trolltech.com.

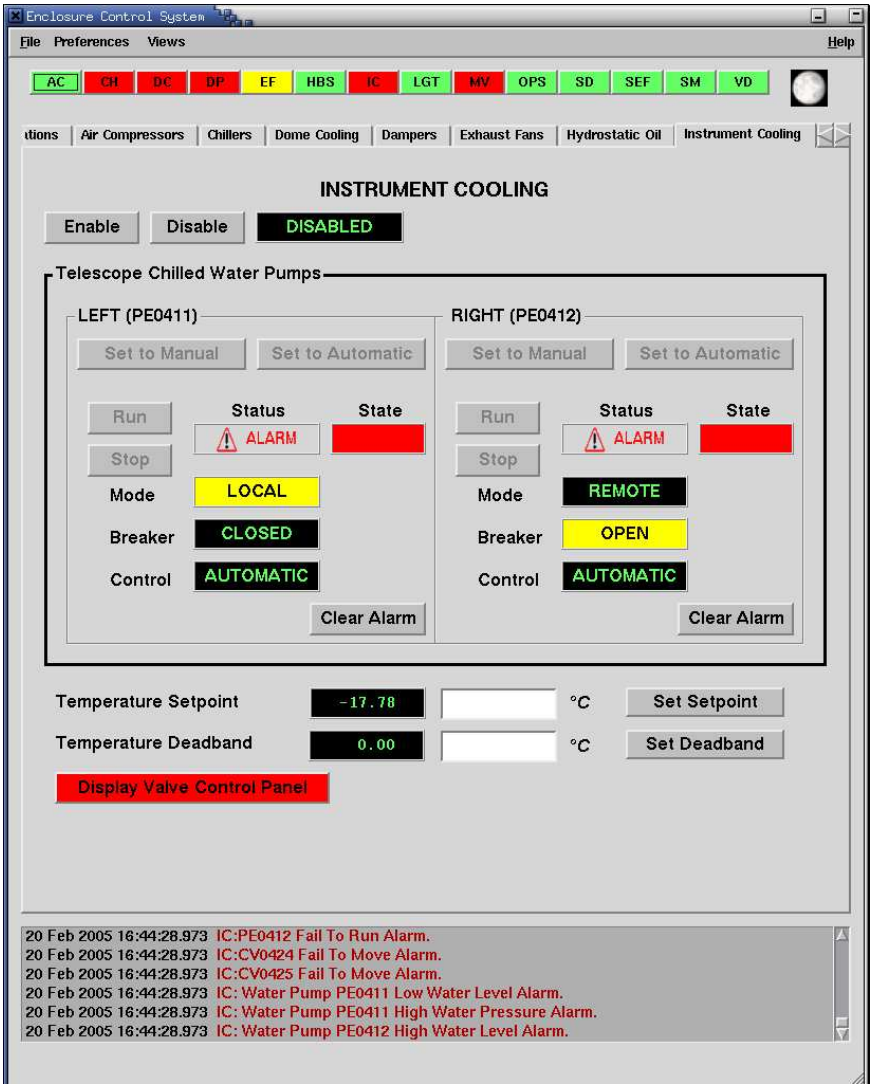

Figure 7: Enclosure Control System, Instrument Cooling main window, full display.

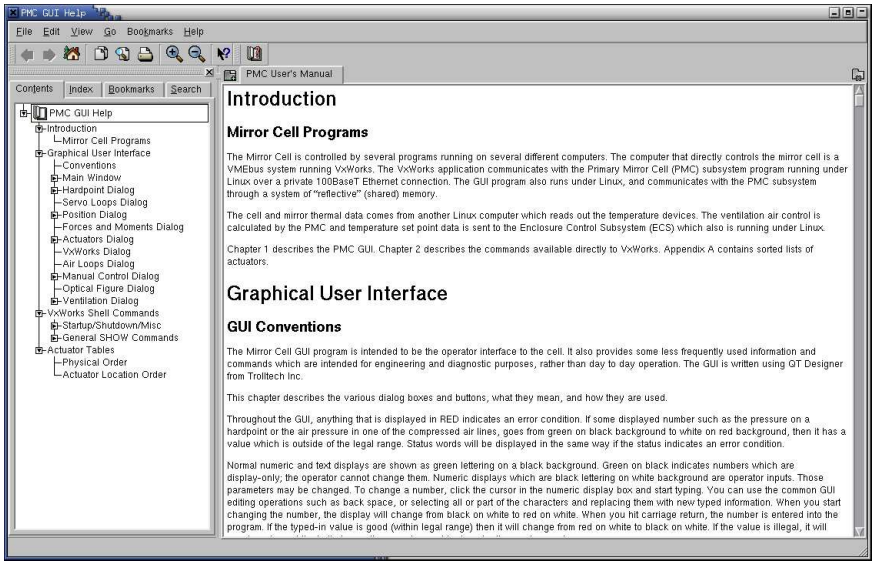

Figure 8: Help dialog for the PMC GUI using Qt Assistant.

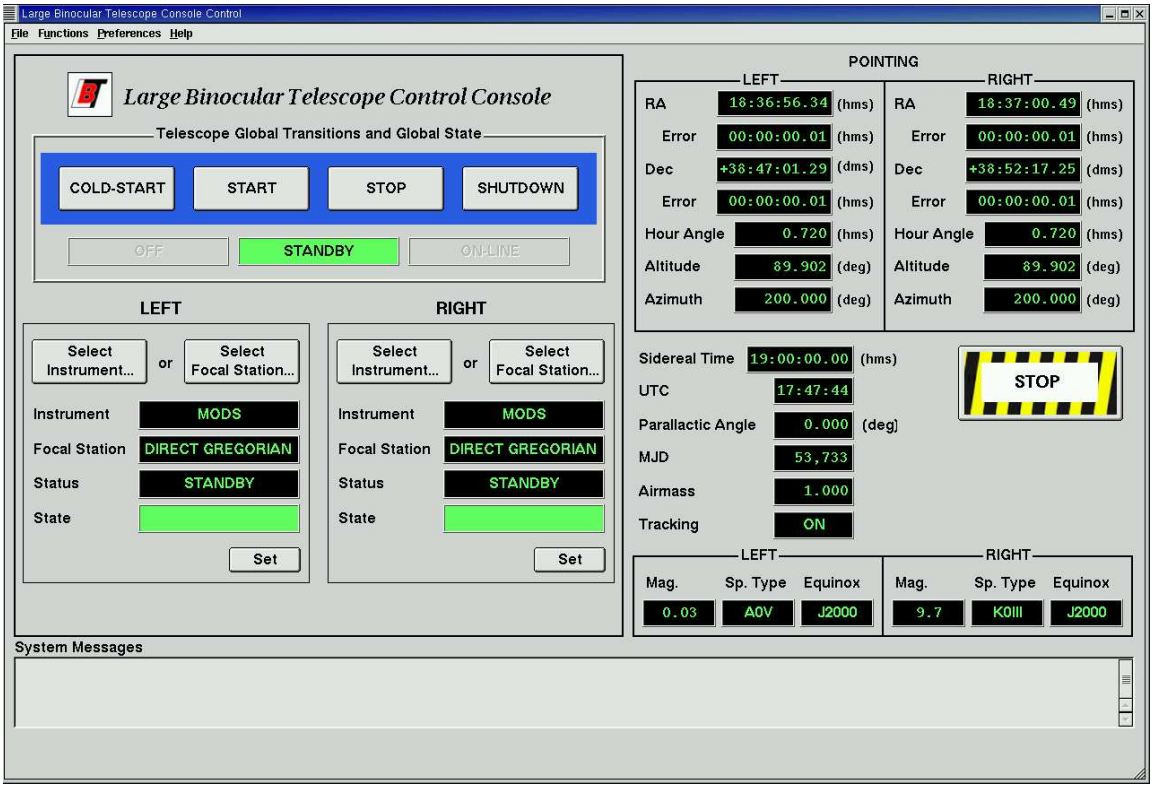

Figure 9: Prototype for the Main Telescope Console main window.