The WIYN telescope graphical user interfaces

Dave Mills

National Optical Astronomy Observatories
950 North Cherry Ave.
Tucson. AZ 85717

March 10, 1995

ABSTRACT

The complexity of modern astronomical instrumentation generates a large amount of status information for the operators to assess. In practice there is far more information than an individual can assimilate. In order to make optimum use of complex systems such as the WIYN active primary mirror system[1] , it is essential that any system events which may impact on the quality of data, be reported in a timely and comprehensible manner.

Using todays powerful workstations we can support the use of real-time visualization of telescope functions and system status. We have developed an interface to the commanding and status layers which will allow us to take advantage of the anticipated future potential for 3D, photorealistic, and virtual display technologies.

1 Introduction

1.1 System architecture

The WIYN telescope control system is a distributed system consisting of a large number of communicating processes running on SunOS and Vx-works hosts. High level I/O to the control system is controlled by a 'router' process. All commands are passed to the router, which in turn passes them to the process which actually executes them. All telemetry is passed to the router which parses it and distributes subsets to 'client' processes. Any process can act as either a client, or a server, or both, and each high level process has a single point of contact with the rest of the system (a router). It is possible to run multiple router processes; for example, a set of client processes at a remote network site could all communicate with the WIYN telescope control system via a single router process (also running at the remote site).

The 'router' interface is provided by a set of library routines with which client programs are linked. Bindings are currently available for C and tcl/tk.[3] The graphical user interfaces described in this paper use the tcl/tk bindings to access the telescope control system (via a router), and thus allow for operation over TCP/IP network connections.

WODC 02-33-01

1.2 Design

1.2.1 Historical perspective

The WIYN telescope system graphical user interface (gui) is based around the pre-existing 4-meter Mayall Telescope gui. The 4-meter gui was implemented using the Sun XView toolkit, and was built using the Sun supplied 'devguide' tools. One of the objectives for the WIYN gui was to provide portable interfaces which can be used at the consortiuum sites to provide 'remote observing'.[2] The WIYN gui is therefore implemented using the portable tcl/Tk toolkits. Some additional tcl/Tk based packages (Tix,tcldp) were also used.

In order to bootstrap the project from the 4-meter gui, a script was written to take the devguide interface description file (xtcs.G), and output a tcl/tk script which provided the same visual appearance.

1.2.2 Implementation

The WIYN gui is designed differently from the 4-meter one, even though its interface appears very similar visually. In particular, most of the widgets do not have any unique code associated with them. Their functionality is controlled through entries in two database files.

- telemetry feedback database
- command sequence database

The database information describing each interface object is of a general nature, and assumes no toolkit specific operations. This design makes it feasible to upgrade the interface to use 3d-shaded or photorealistic representations in future without changing either the interface logic code, or the databases. All that would be required are a few toolkit interface routines which provide generic operations on the new interface components (eg show, hide, set). The addition of new generic operation types in the future is simplified by the fact that only two routines refer to them; one which reads the database, and the second which calls the toolkit routine requesting the action be performed on an interface object. The following text refers to the current tcl/tk toolkit based implementation.

For example, to configure a button widget to send a sequence of telescope commands whenever the user presses the button, the following tk code would be required

```
.some.widget.name configure -command 'cmdsequence sequence-name'
```

The 'cmdsequence' routine provides a generic entry point for all commanding, and parses the command sequence database to obtain the correct sequence of commands. The visual feedback which indicates the current telescope status, and reflects changes in that status due to commanding (or any other cause), is controlled via entries in the telemetry feedback database. The two databases are logically separated because it is entirely possible for the telescope state to change in an arbitrary manner, independent of commands sent via the gui (either because of cli commanding, or hardware failures). The feedback database allows us to link widget configuration actions such as color changes, displayed text/value, or visibility, with any required telemetry state, without writing any widget specific code.

1.3 Databases

1.3.1 The command sequence database

The interaction of the WIYN gui with the telescope commanding is controlled primarily through entries in the command sequence database. The database is currently stored in ascii format. The database contains a set of named command sequences, each of which may consist of an arbitrary number of commands. Command sequences may be linked to any number of widgets. The linkage is made by setting the widget up to invoke 'cmdsequence name', where cmdsequence is a generic procedure which extracts the correct sequence, makes any necessary variable substitutions, and submits the commands.

Command sequences are initiated in two different ways in the WIYN gui.

- Immediate sequences are initiated by pressing a single button on the gui and proceed to transmit all their commands, unless blocked by telemetry conditions, or cancelled by the operator.
- Queued sequences are constructed by the operator by selecting multiple options from 'Inspector panes'. No commands are actually sent until the operator presses the 'apply' button, although the queued commands may be examined in the command history window.

Each command sequence consists of a single header line which contains the sequence name plus any hint text. if a hint text is provided then it will be copied to the command history when the command sequence is invoked. The header is followed by an arbitrary number of action lines. The action lines have one of the following formats

- waitfor telemetry-parameter condition value
- waitfor nnn seconds
- system subsystem component state [state2] [...]

The 'waitfor' forms are parsed and interpreted by the command sequencer. All other commands are passed to the telescope control system for execution. In the WIYN implementation, the tcs command interpreter is actually a tcl interpreter so we can include tcl commands in our sequences (in practice, use of this is limited to 'source'ing other script files). Apart from the self-explanatory 'waitfor 5 seconds' form, the 'waitfor' conditions are expressed using the same syntax as conditions in the telemetry feedback database.

e.g. `waitfor dcs.dome.ccwmotion = 0`

would cause the command queue to block until the telemetry indicated that the required condition had been reached.

1.3.2 The telemetry feedback database

The interaction of the WIYN gui with the telescope telemetry is controlled via entries in the feedback database file. This file is in ascii format and is parsed on startup by the main tcs gui. It contains entries for

- Setting hint texts
- Links to the command sequence database
- Action items to perform depending on telemetry values

The following widget types are supported

- Window - introduces a collection of widgets
- Button - a single button
- Menubutton - a pull-down menu button
- Led - a state indicator
- Choice - a set of radiobuttons
- Value - an alphanumeric user-editable field

A widget descriptor record has the following format

```
type name [hint-text]
```

e.g. `button main_rotators Enable/Disable rotator tracking`

The action descriptor records provide the linkage between telemetry values and actions in the gui. Typically a telemetry value may cause a widget to be modified in one of the following ways

- Color - change the widget foreground color
- Show - make the widget visible
- Hide - make the widget invisible
- Set - set a choice widget, or a value widget
- Update - connect a value widget to a command sequence item

The action descriptor record has the following format

```
action parameter condition modifier [modifier2]
```

e.g. `set tcs.dome.encoderstate = absolute init`

In the above example, the action would be associated with a choice widget. When the telemetered value of 'tcs.dome.encoderstate' is equal to the value 'absolute', then the 'init' button is set active, and the other choice components are set inactive. Each widget descriptor may have an arbitrary set of action descriptors associated with it. For 'value' widgets only, the widget may have an 'update' action descriptor. This descriptor has the following format

```
update command-sequence-id [min value max value]
```

e.g. `update c_targetspeedx min 0 max 10000`

This action descriptor links the value widget to the command sequence 'c_targetspeedx'. Whenever the operator changes the value of the field, it will be limit checked (if limits were provided), and the value of the field will be substituted into the command sequence by replacing every occurence of '$widgetname' with the new value.

Using the gui 'Inspector panes', it is possible for the operator to queue up different sets of commands which may be mutually exclusive. For example they may queue an 'active secondary enable', and then an 'active secondary disable', before pressing the 'apply' button. Rather than providing code to deal with each possible case, each command sequence is assigned a numbered slot, and mutually exclusive sequences overwrite the same numbered slot, thus ensuring that the latest sequence queued will be the only one that gets executed (for each particular slot).

The priority works in a similar fashion to ensure that sequences which must be performed before others (eg 'active secondary enable' before 'active secondary focus adjust'), are , without having to write code for each case.

2 Graphical user interfaces

2.1 Telescope control

The main window (figure 1) is used to initiate the most common operations used during observing, such as slewing the telescope. The main window also has a set of menus and buttons which provide easy access the other facilities such as the GSC guide star catalog, object oracle, command history etc. Buttons which open extra windows are configured as toggle buttons, so that the first click opens the window, and the next click hides it again.

All aspects of the interface provide 'hints' to the operator. When the mouse passes over an active area of the interface (eg a button), the hint for this item appears at the bottom of the window. Context sensitive help is also provided for a more detailed description of the interface. Pressing the 'Help' key on the keyboard whilst the mouse is positioned over the item of interest will display the detailed help for that item.

2.1.1 Subsystem inspectors

The gui provides a number of 'Inspectors' (figure 3), each of which provides information and control for an individual subsystem. For example the secondary mirror subsystem has its own inspector window. Some commonly used inspectors are available directly from buttons on the main window (eg Focus inspector).

2.1.2 Command history

The command history window (figure 2) is made visible/hidden using the history button in the main window. It contains reports on all commands and their status. For example, commands may be queued awaiting execution. Each command is accompanied by a time/date tag. The most recent command is always at the top of the window.

2.1.3 Open/stow window

The Open/Stow window (figure 4) is used during initial setup, and to close down. It contains all the functions necessary to begin telescope operations at the beginning of an observing run. It is made visible/hidden using the 'Open/Stow' button in the main window.

2.2 Observing tools

2.2.1 Next object selection

The next object (figure 6) window allows the user to manually input the coordinates for the next object to be observed. Clicking 'apply' makes active the selection. It is also possible to search the FK5 catalog for a star close the specified Alt/Az coordinates.

2.2.2 GSC catalog search

This window controls the GSC catalog access (figure 5). It is opened when the 'gstar next' button is clicked in the main window and has two modes of operation. Clicking 'gstar next' performs a GSC catalog search centered on the current 'next object'. It returns the closest match to the main window as a guide star associated with the 'next object'. Once the 'gstar' window is visible, the user may also manually initiate searches, and also view the full set of objects returned from a search and manually select one (by clicking on 'list' in the gstar window).

If the interface is configured to automatically assign GSC guide stars ('auto gstar' in the main window), then the 'gstar' window will remain hidden and guide stars will be automatically assigned whenever a new position is entered into the 'next object' box in the main window.

2.2.3 Object oracle window

The object oracle (figure 7) provides the operator with a graphical indication of the time remaining until certain critical events, such as 1.5 airmasses, 2 airmasses, elevation less than 15 degrees, etc. The oracle is initiated from the 'main' window and once visible is automatically updated whenever the telescope is slewed to a new position. The operator may also manually retrieve previous object graphs, or enter a new object position directly into the oracle window.

2.2.4 GSC skymap window

The GSC skymap window is opened by clicking on the button in the main window. Once open it will automatically update every time the telescope is slewed to a new position. Clicking on an object in the skymap will open an information window for the object, and also provides a 'send to telescope' function which makes it the 'next object' GSC skymap also provides access to objects in HD,YBS,SAO, and RNGC catalogs.

2.3 System status monitors

2.3.1 Telemetry - ascii

The main telemetry monitor is table driven and provides easy customization of displayed telemetry. Telemetry items can be grouped under header button's which act as toggles to show/hide that group of items. Each telemetry item may be displayed either as ascii text, or as an icon (chosen from a set based upon the current item value). Alarm monitors may also be configured such that the window containing the item automatically opens whenever the item changes its value (the changed item is also highlighted before changing to the state representing its new value).

2.3.2 Primary mirror supports and thermal environment

The primary mirror support window (figure 8) presents a graphical representation of the 66 axial and 4 lateral active supports. Selections may be made to display the active forces, the lookup table entries, or the current forces. The display is color-coded to give a broad overview of the support system status. Clicking on an individual actuator provides access to more detailed information about that unit, including a historical commentary of any observed problems.

2.3.3 Telemetry - graphical

The telemetry plotter (figure 9) is started from the main window by selecting the 'graphics' entry on the 'telemetry' menu. Once started it provides access to any item in the telemetry. Items may be plotted against each other or against time. Any area of a graph may be examined in detail by using the mouse to drag an outline surrounding the area.

Up to four different graphs may be active simultaneously and each may be dismissed, hidden,or reset. Graphs are updated whenever telemetry arrives (usually 1Hz) and the data may be printed or saved to disk at any time. Limited statistics are also available by clicking the 'rms' button.

3 Summary

The data-centric design provides flexibility and expandability. The implementation is easy to modify and the underlying code contains no WIYN telescope specific knowledge. The current system is layered on top of freely available packages (tcl/tk) which are portable to a wide range of operating systems.

4 Figures

4.1 Figure 1 - The main telescope control window

4.2 Figure 2 - Command History window

4.3 Figure 3 - A subsystem inspector window

4.4 Figure 4 - Open/Stow window

4.5 Figure 5 - GSC CDROM search control window

4.6 Figure 6 - Next object input window

4.7 Figure 7 - The Object 'oracle' window

4.8 Figure 8 - Primary mirror active support monitor

4.9 Figure 9 - Graphical telemetry monitor

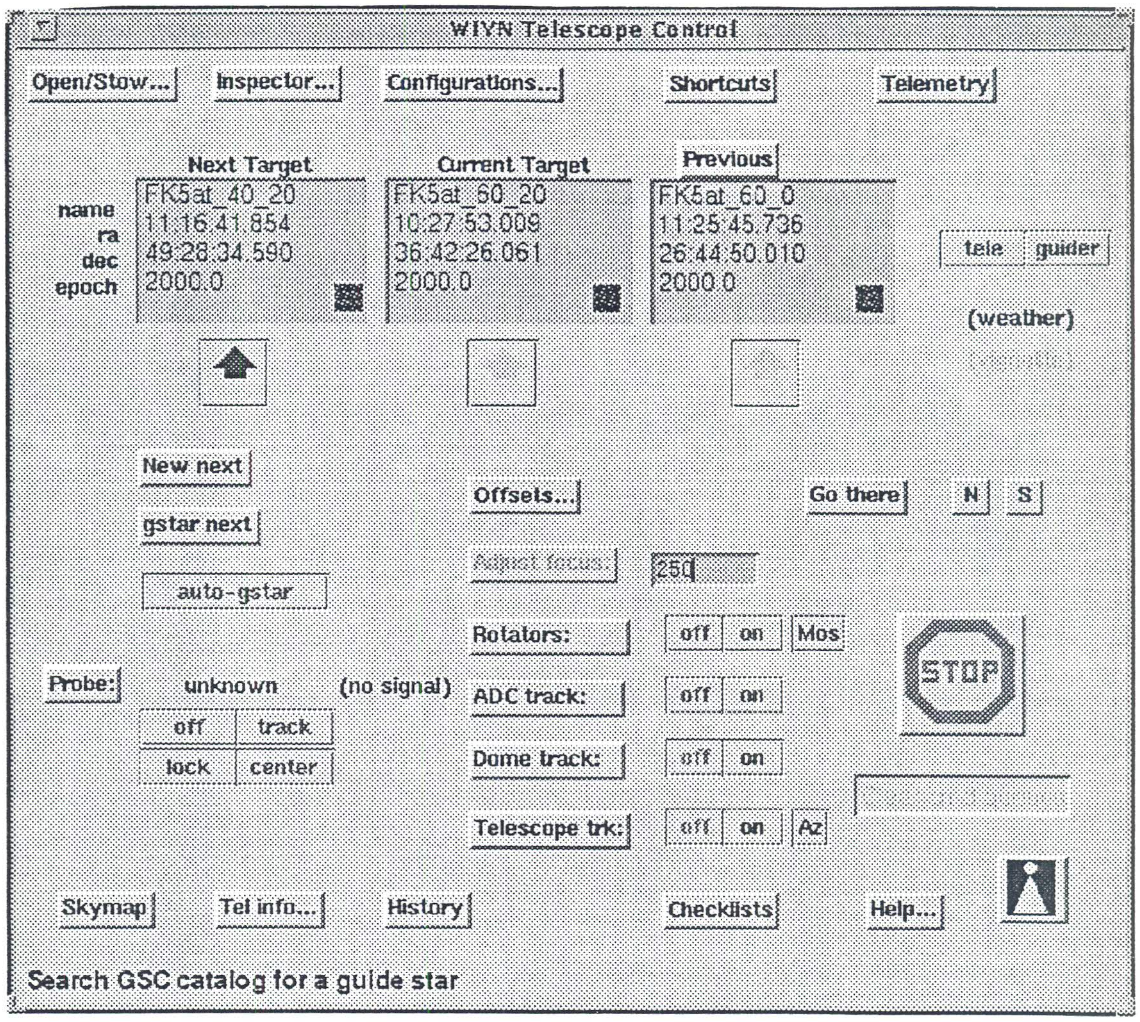
5 REFERENCES

[1] N. Roddier, L.W. Goble, D.R. Blanco, C.A. Roddier.
WIYN active optics system
these proceedings 2479-26

[2] J.W. Percival.
Remote Observing from the bottom up: the architecture of the WIYN telescope control system.
these proceedings 2479-05

[3] J.K. Ousterhout.
Tcl and the Tk toolkit
Addison Wesley (ISBN 0-201-63337-X) 1994

Figures



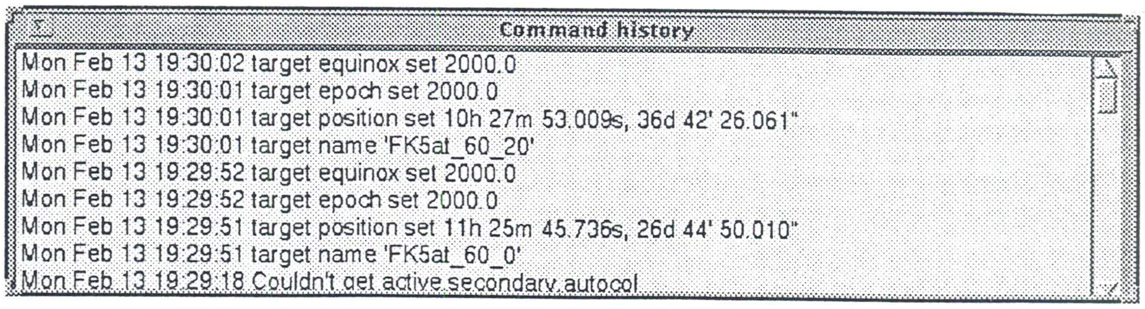
	Next Target	Current Target	Previous
name	FK5at_40_20	FK5at_60_20	FK5at_60_0
ra	11:16:41.854	10:27:53.009	11:25:45.736
dec	49:28:34.590	36:42:26.061	26:44:50.010
epoch	2000.0	2000.0	2000.0

Adjust focus: 250

Probe: unknown (no signal) — off, track, lock, center

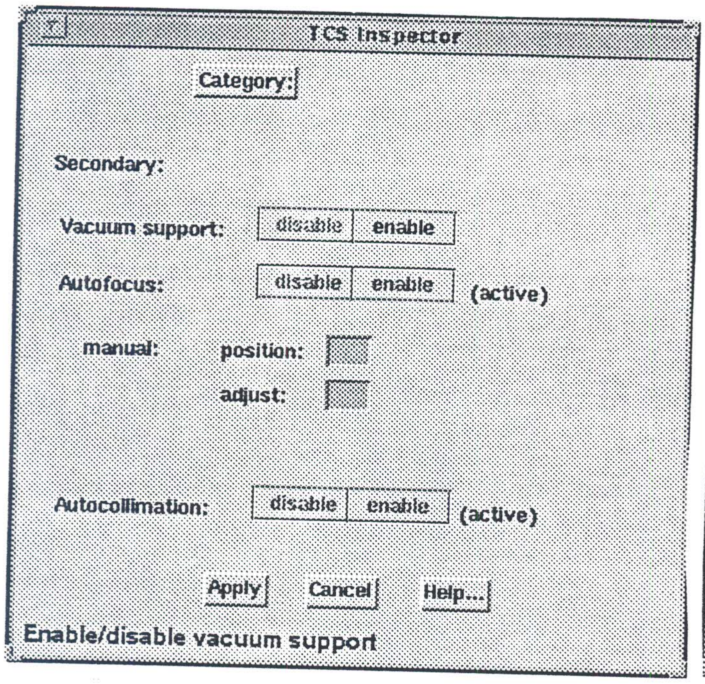
Search GSC catalog for a guide star

Figure 1 - The main telescope control window



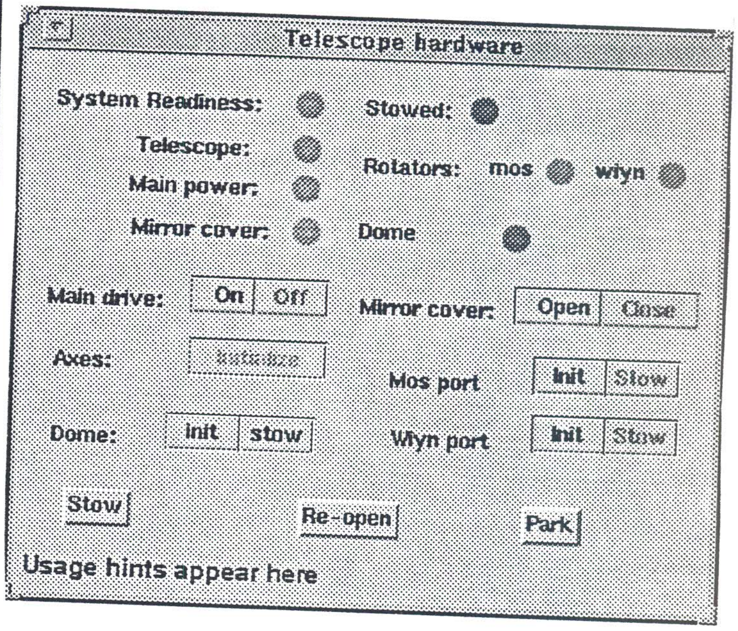
```
Mon Feb 13 19:30:02 target equinox set 2000.0
Mon Feb 13 19:30:01 target epoch set 2000.0
Mon Feb 13 19:30:01 target position set 10h 27m 53.009s, 36d 42' 26.061"
Mon Feb 13 19:30:01 target name 'FK5at_60_20'
Mon Feb 13 19:29:52 target equinox set 2000.0
Mon Feb 13 19:29:52 target epoch set 2000.0
Mon Feb 13 19:29:51 target position set 11h 25m 45.736s, 26d 44' 50.010"
Mon Feb 13 19:29:51 target name 'FK5at_60_0'
Mon Feb 13 19:29:18 Couldn't get active secondary autocol
```

Figure 2 - Command History window


TCS Inspector
Category:
Secondary:
Vacuum support: disable enable
Autofocus: disable enable (active)
manual: position:
adjust:
Autocollimation: disable enable (active)
Apply
Cancel
Help...

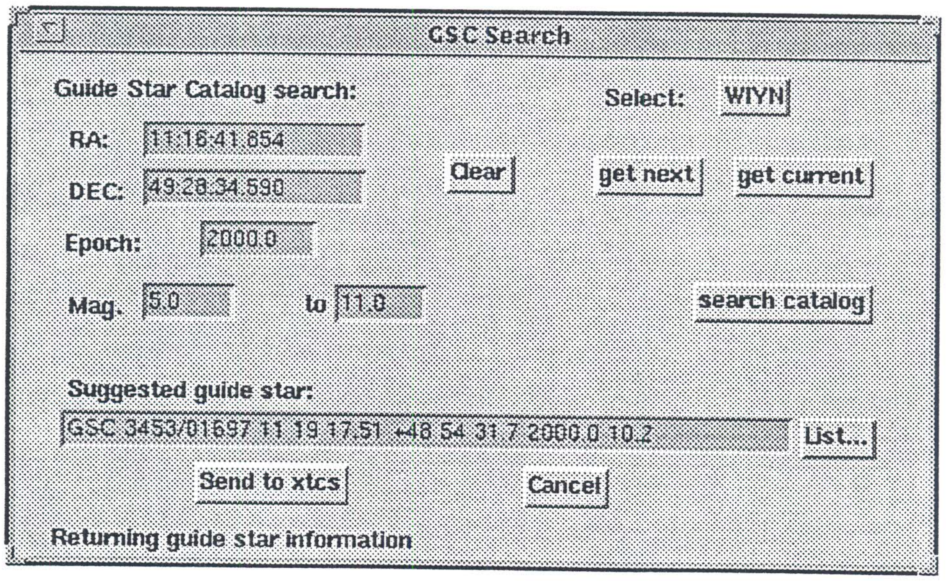
Enable/disable vacuum support

Figure 3 - A subsystem inspector window



Usage hints appear here

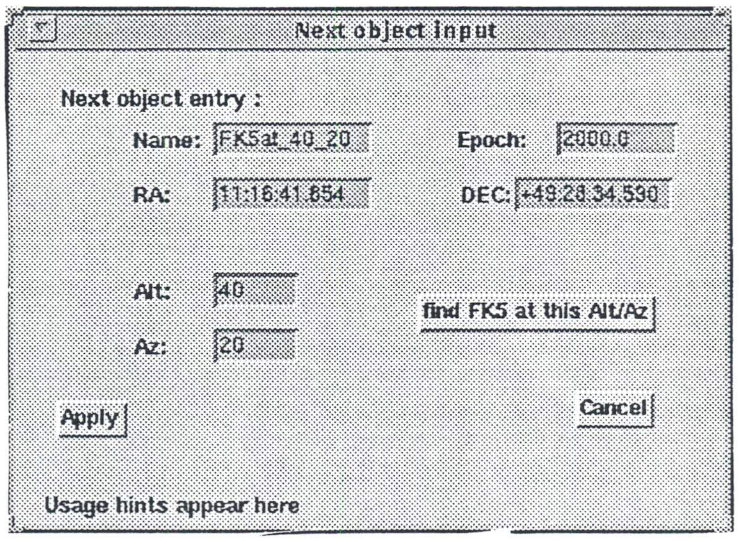
Figure 4 - Open/Stow window



RA: 11:16:41.654
DEC: 49:28:34.590
Epoch: 2000.0
Mag. 5.0 to 11.0
GSC 3453/01697 11 19 17.51 +48 54 31.7 2000.0 10.2

Returning guide star information

Figure 5 - GSC CDROM search control window


Next object input

Next object entry :

Name: FK5at_40_20 Epoch: 2000.0

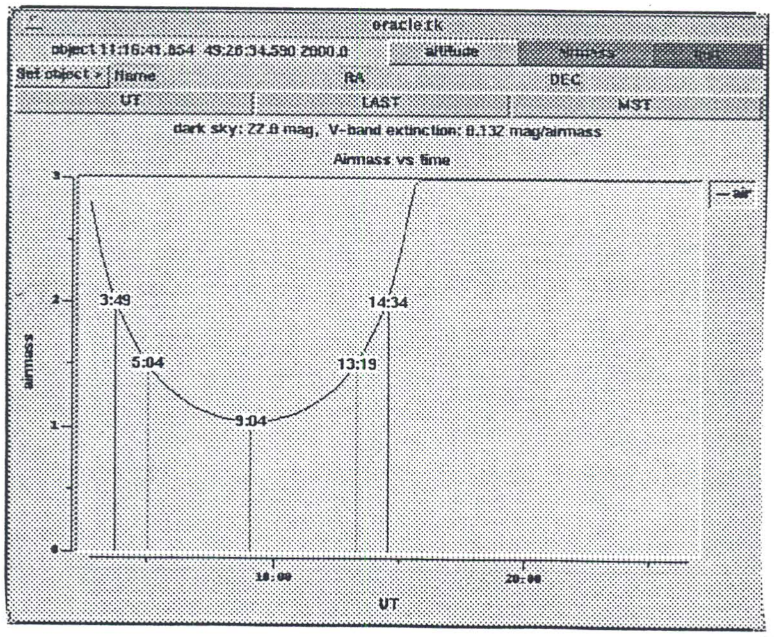
RA: 11:16:41.654 DEC: +49:28:34.590

Alt: 40

Az: 20

Usage hints appear here

Figure 6 - Next object input window


oracle.tk

object 11:16:41.854 49:26:34.590 2000.0

dark sky: 22.0 mag, V-band extinction: 0.132 mag/airmass

Figure 7 - The Object 'oracle' window

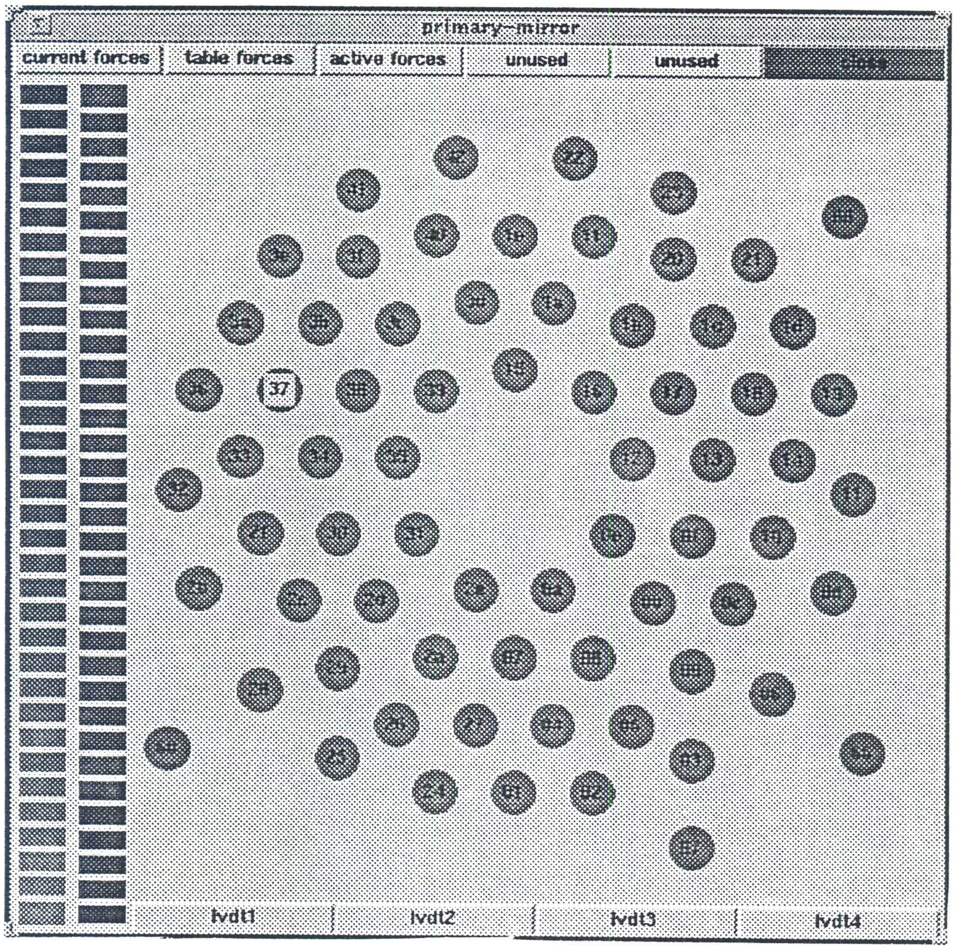


Figure 8 - Primary mirror active support monitor

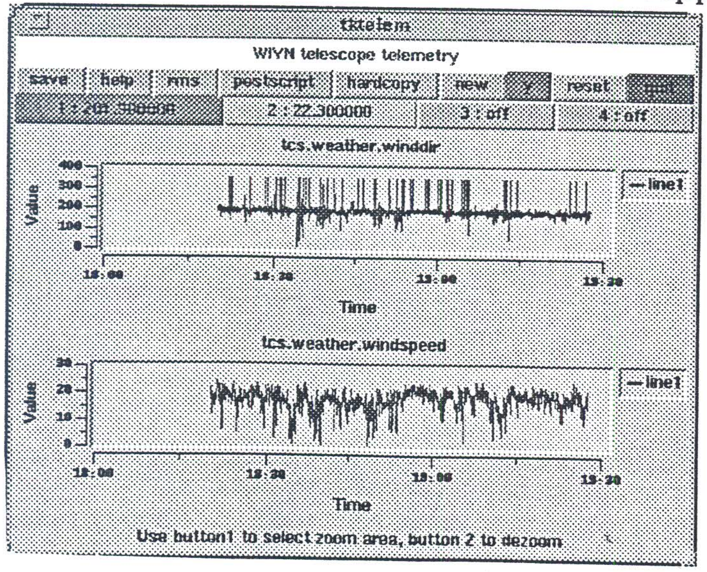
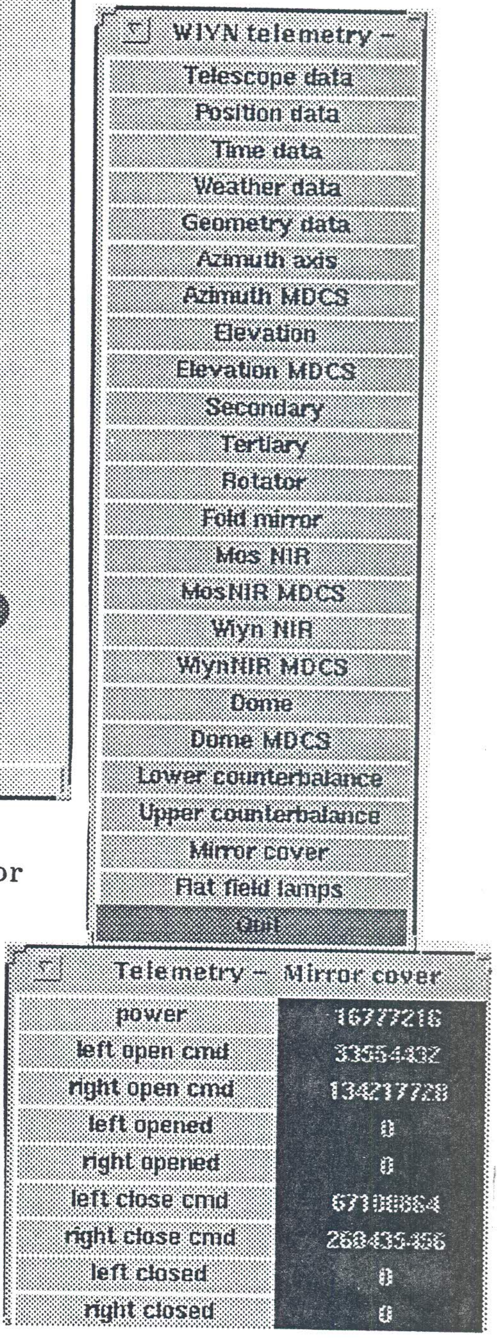


Figure 9 - Graphical telemetry monitor



power	16777216
left open cmd	33554432
right open cmd	134217728
left opened	0
right opened	0
left close cmd	67108864
right close cmd	268435456
left closed	0
right closed	0

Figure 10 - ASCII telemetry monitor