

---

## NIRC2 Software Design Book

File: 3.5.6 (KSDs), Library KSD Binder, Summit KSD Binder

Circulation:

### Table of Contents

1.0	Introduction.....	4
1.1	Brief History.....	4
1.1.1	Software CDR (October 1997) .....	4
1.1.2	NIRC1 P3 Software Project (August 1997 - February 1998).....	4
1.1.3	UCLA/NIRSPEC Readout Electronics Decision (April 1998) .....	4
2.0	Review of the 1997 Board Report.....	5
2.1	Design Flows from Requirements.....	5
2.2	Design Avoids Complexity and Duplication.....	5
2.3	Realistic Schedule and Demonstrable Milestones .....	5
2.4	Modular .....	5
3.0	Architecture .....	6
3.1	Command Line Interface and GUIs .....	7
3.1.1	Command Line Interface .....	7
3.1.2	Command Script Examples .....	11
3.1.3	Graphical User Interface.....	14
3.2	Mechanism Keyword Library.....	16
3.2.1	Observing keywords .....	18
3.2.2	Maintenance keywords .....	22
3.2.3	Engineering keywords .....	22
3.2.4	General purpose motor keywords .....	23
3.2.5	Moving motor behavior .....	24
3.2.6	Configuration files.....	25
3.2.7	Logging .....	28
3.2.8	Keyword-changed events/notifications .....	28
3.2.9	Simulation .....	28
3.3	Motor daemon .....	30
3.3.1	Rpc functions .....	31
3.3.2	Configuration files.....	32
3.3.3	Support programs.....	35
3.3.4	Logging .....	36
3.3.5	Simulation .....	36

---

3.3.6	Motor controllers .....	36
3.4	I/O Daemon .....	38
3.4.1	Rpc functions .....	38
3.4.2	Configuration files.....	41
3.4.3	Support programs.....	43
3.4.4	Logging .....	44
3.4.5	I/O Device controllers.....	44
3.4.6	Temperature Control .....	45
3.4.7	Coldhead Control .....	45
3.5	Pupil Tracking Control .....	45
3.6	UCLA/NIRSPEC Software for Reading the Detector .....	47
3.6.1	Keywords .....	48
3.6.2	Quick Look .....	49
3.6.3	Occam Transputer Software.....	50
3.6.4	Subarray Clocking .....	50
3.6.5	ALADDIN 3 Modifications .....	50
3.6.6	FITS Writer Modifications.....	50
4.0	Software Environment .....	52
4.1	Desktop.....	52
4.2	Logging and alarms .....	53
4.2.1	Thermal Logging .....	53
5.0	Software Test Plan .....	53
5.1	Testing the LAB I&T Release .....	54
5.1.1	Testing Motor Keywords .....	54
5.1.2	Testing DGH Keywords .....	54
5.1.3	Testing Lakeshore Keywords.....	54
5.1.4	Testing ALADDIN 3 Keywords .....	54
5.1.5	Scripts .....	54
5.2	Testing the Complete Release .....	54
6.0	Schedule.....	54
Appendix A.	Design Goals .....	58
A.1	Simplicity .....	58
A.2	Modularity.....	58
A.3	Reliability .....	58
A.4	Testability .....	59
A.5	Extendibility .....	59
A.6	Error Checking .....	59
Appendix B.	Requirements Cross Reference .....	60
Appendix C.	Proposed NIRC2 Keywords .....	69
Appendix D.	KTL Keywords .....	79
D.1	Figures .....	79
D.2	Shell Level Commands.....	81
D.3	C Calling Sequences .....	81
D.4	Example Commands .....	81

---

Appendix E. Glossary ..... 83  
Appendix F. List of Figures ..... 84  
Appendix G. List of Tables ..... 85

## 1.0 Introduction

This document presents a software design for NIRC2, a second generation near infrared camera for the Keck Telescope. The design is based on the NIRC2 Software Functional Requirements (NIRC2-98001).

The detector electronics and readout software provided by UCLA (essentially the NIRSPEC system) are not described here, except to the extent that we will modify and interface with this system to apply it to NIRC2.

### 1.1 Brief History

In this section, we review episodes of NIRC1 and NIRC2 history that affect the design described here.

#### 1.1.1 Software CDR (October 1997)

The design presented at the NIRC2 Software CDR on 28-29 October 1997 was not accepted by the review board. In its report, the review board noted that the proposed design

- did not flow from requirements;
- was unnecessarily complex and duplicated functionality available within existing operating systems and other available software;
- depended on an unrealistic schedule;
- did not allow phased completion through demonstrable milestones; and
- could not be modularized to allow application of additional manpower.

#### 1.1.2 NIRC1 P3 Software Project (August 1997 - February 1998)

Between August 1997 and February 1998, a CARA team implemented a software system for the first generation near-infrared camera (NIRC1). This system, sometimes referred to as the "P3" system, shares the following attributes with the currently proposed NIRC2 design:

- Both systems are command line based.
- Both make extensive use of unix csh and RSI IDL.
- In both cases, new software for motor control is required.
- In both cases, the software for reading the detector and displaying images is inherited.
- The cameras are similar to one another.
- Considerable overlap exists in the user communities for the two cameras.

#### 1.1.3 UCLA/NIRSPEC Readout Electronics Decision (April 1998)

In April 1998 a decision was made to adopt UCLA/NIRSPEC electronics and software for use on NIRC2. This decision was made with the assumption that it would be used "as is," with little or no change. In particular, data rate requirements to support speckle were removed.

## 2.0 Review of the 1997 Board Report

In this section we describe how the current design addresses the points made in §1.1.2, above. A more extensive discussion of design goals is given in Appendix A.

### 2.1 Design Flows from Requirements

To be sure that this design flows from requirements we

- carefully reviewed requirements with the PI and CARA instrument scientists, and
- produced a cross reference (Appendix B) that, for each requirement given in NIRC2-980001, specifies which sections in this document address that requirement.

### 2.2 Design Avoids Complexity and Duplication

To be sure this design is not unnecessarily complex and does not duplicate functionality available within existing operating systems we

- Inherit substantial components from P3 (§1.1.2) including:
  - use csh instead of a custom CLI (§3.1)
  - use IDL instead of a custom math package (§3.1)
  - inherit most of the P3 keyword library code (§3.2)
  - inherit most of the P3 motor and I/O daemon code (§3.3 and §3.4)
  - inherit the syslog/tklogger mechanism used on existing Keck instruments (§4.2)
- Use Sun's RPC package for interprocess communication.
- Inherit the detector readout software from UCLA/NIRSPEC (§1.1.3 and §3.6).

### 2.3 Realistic Schedule and Demonstrable Milestones

We have produced a schedule with demonstrable milestones (§6.0).

### 2.4 Modular

To ensure this design is modular and will, if necessary, allow application of additional manpower, we

- Inherit substantial code from existing systems (§2.2). Any of the staff familiar with those systems can quickly be pulled into the project if a speed up becomes necessary.
- Ensure that individual modules (Figure 1) are well defined.

### 3.0 Architecture

Figure 1 provides a top level view of the NIRC2 software design and also serves as a road map to the detailed design presented in this section.

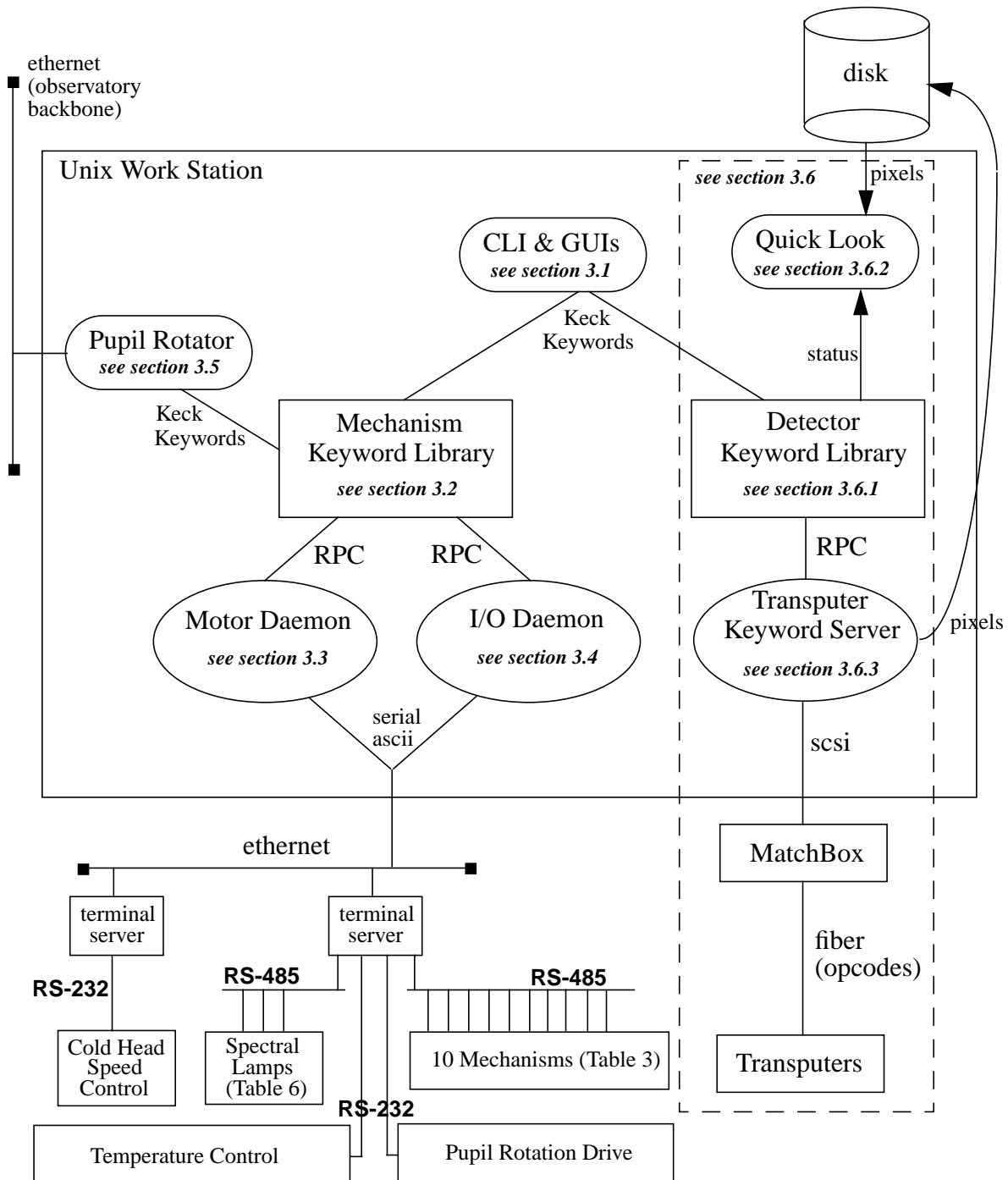


FIGURE 1. NIRC2 Software Overview.

The UCLA and CARA software will all reside on a single Unix work station. The design provides, however, for moving any of the motor daemon, the I/O daemon, or the Transputer Keyword Server, to any Unix work station on the same subnet.

Top level clients, such as scripts and GUIs interface to lower level servers via the Keck Tasking Library (KTL, Appendix D), and the shareable keyword libraries that KTL activates. The mechanism keyword library will use Remote Procedure Calls (RPC) to communicate with motor and I/O daemons which will handle device-specific details and the serial communications with those devices. Similarly, the Detector Keyword Library will use RPC to communicate with the Transputer Keyword Server.

The mechanism and detector keyword libraries will be developed independently at two different sites. Keeping these libraries separate from one another eases integration considerably, and, from the point of view of the observer, will be transparent.

## 3.1 Command Line Interface and GUIs

### 3.1.1 Command Line Interface

This section describes commands available to the observer. Most commands will be written as Unix scripts. Using Unix as the operating system has the great advantage of a large suite of stable, robust commands and features, from simple file renaming commands like “mv” to more sophisticated attributes like trapping Control-C interrupts for customized processing. Many, if not most, observers also have significant prior Unix experience, easing the “learning curve.”

Scripts are used to (a) protect the user from having to know low-level details about the system (such as the position in millimeters of a slit slide required to put a given slit at a given position on the detector), and (b) automate natural sequences of commands. To demonstrate the latter, consider a dither pattern, which consists of a sequence of telescope moves interspersed with images. A simple command such as “bxy9 5” can be used to automate a nine-point dither pattern with a leg size of 5 arcseconds (the parameter passed to the script). This could replace 18 single commands (nine telescope moves and nine images).

Many commands will be found in common with NIRC1 (Table 1). This is, by design, again to retain and build upon the existing observer knowledge of how to run NIRC1. I/O in the window provided for user commands will be logged to a file.

TABLE 1. NIRC2 shell scripts and their functions.

Procedure	Units	Description	Requirements section
<b>help</b> [ <i>command</i> ]	text	Display a brief list of commands, or more detailed information on <i>command</i> .	3.1.9
<b>abs</b> <i>x</i>	float	Take the absolute value of <i>x</i> .	
<b>ampang</b> <i>x1 y1 x2 y2</i>	float	Converts two (x,y) pairs into a distance and an angle. The (x,y) coordinates are assumed to come from NIRC2, hence a scale factor appropriate to the camera in use is used.	
<b>ao2flat</b>		Change AO deformable mirror to flat.	6.1.11.12
<b>ao2laser</b>		Change AO mode to use laser guide star.	6.1.11.12
<b>ao2nat</b>		Change AO mode to use natural guide star.	6.1.11.12

<b>ao2real</b>		Turn off AO simulator and begin talking to the real AO system.	6.1.11.12
<b>ao2sim</b>		Turn on AO simulator.	6.1.11.12
<b>autodisp</b>		Switch the display intensity mapping to “auto”.	
<b>az a</b>	arcsec	Move telescope <i>a</i> arcsecs in azimuth.	6.1.11.2
<b>azel a e</b>	arcsec	Move telescope <i>a</i> arcsec in azimuth, <i>e</i> arcsec in elevation.	6.1.11.2
<b>bells n</b>	integer	Ring the terminal bell <i>n</i> times.	
<b>box5</b>		Move telescope in a 5-point box pattern and take frames.	6.1.12.1
<b>box9</b>		Move telescope in a 9-point box pattern.	6.1.12.1
<b>bstat [n]</b>	integer	Calculates statistics (mean, standard deviation, median, etc.) on buffer <i>n</i> (buffer 1 if <i>n</i> is not specified).	5.3.2.4, 5.3.2.5
<b>buf1 = buf2 [+/- buf3]</b>		Buffer copy or, if an operator and third buffer are specified, buffer math. There will be commands for buf1 through buf4.	3.1.5, 5.3.2.1, 5.3.2.2
<b>bxy5 x y</b>	arcsec	Move telescope in a 5-point pattern aligned with the detector.	6.1.12.1
<b>bxy8 x y</b>	arcsec	Move telescope in an 8-point pattern aligned with the detector.	6.1.12.1
<b>bxy9 x y</b>	arcsec	Move telescope in a 9-point pattern aligned with the detector.	6.1.12.1
<b>camera x</b>	text or float	Select camera <i>x</i> , where <i>x</i> can be either a camera number (1–3) or a scale (e.g. “0.01”).	6.1.4.1
<b>cdata</b>		Shows the files in the current data directory.	6.1.13.1
<b>cent x y</b>	pixels	Move an object at ( <i>x</i> , <i>y</i> ) to the center of the detector.	
<b>clrgrism</b>		Clears the grism slide.	6.1.5.3
<b>close</b>		Close the slits and mask.	6.1.6.4
<b>closeon text</b>	text	Tells the shutter to close on a given criterion (specified by <i>text</i> ) received from the AO system (such as Strehl ratio below some value, etc.).	6.1.11.16
<b>coadd n</b>	integer	Set the number of coadds to <i>n</i> .	6.1.1.2.4
<b>corona n y</b>	integer, float	Set the coronagraphic spot <i>n</i> to position <i>y</i> .	6.1.9.1
<b>dark</b>		Inserts the aluminum cold plug (used for bias frames).	
<b>dekker n</b>	integer	Read the slit position (parametrized by the mask position) and insert dekker <i>n</i> at that position.	6.1.6.2
<b>detbias n</b>	integer	Set the detector bias.	6.1.1.2.4
<b>dfoc df</b>	float	Change the telescope focus by a differential of <i>df</i> mm.	6.1.11.7
<b>dir</b>		List data files.	6.1.13.1
<b>dp n</b>	integer	Display picture <i>n</i> .	
<b>dpname text</b>	text	Display file <i>text</i> .	
<b>e e</b>	arcsec	Move telescope east <i>e</i> arcsec.	6.1.11.1
<b>el el</b>	arcsec	Move telescope in elevation <i>el</i> arcsec.	6.1.11.2
<b>en e n</b>	arcsec	Move telescope east <i>e</i> arcsec, and north <i>n</i> arcsec.	6.1.11.1
<b>endnight</b>		Shutdown; does <b>dark</b> , <b>close</b> , and turns off electronics.	
<b>fgr grism filt</b>	integer, text	Select a grism located in a filter wheel and optionally insert blocking filter.	6.1.3.3, 6.1.3.4
<b>filter m n</b>	integer	Movethe first filter wheel to position <i>m</i> , the second to position <i>n</i> .	6.1.3.1
<b>foc f</b>	mm	Set telescope focus to <i>f</i> .	6.1.11.6
<b>foc3 df</b>	mm	Step through three focus positions, incrementing focus by <i>df</i> mm each time. Display an image constructed from the original images.	6.1.12.3
<b>foc5 df</b>	mm	Step through five focus positions, incrementing focus by <i>df</i> mm each time. Display an image constructed from the original images.	6.1.12.3

<b>foc8</b> <i>df</i>	mm	Step through eight focus positions, incrementing focus by <i>df</i> mm each time. Display an image constructed from the original images.	6.1.12.3
<b>frame</b> <i>n</i>	integer	Set the next frame number to <i>n</i> .	6.1.1.2.2
<b>fromsky</b>		Offset the negative of the telescope by the nod parameters.	
<b>go</b>		Prompt for taking an exposure.	6.1.1.1.2
<b>goi</b> [ <i>n</i> ]	integer	Take an exposure. If <i>n</i> is specified, <i>n</i> exposures will be taken	6.1.1.1.1
<b>gomark</b>		Return to the previously stored telescope offsets (see <b>mark</b> ).	
<b>grand</b>		Output a random number.	
<b>gxy</b> <i>x y</i>	arcsec	Move telescope in guider coordinates.	
<b>h</b>		Insert the H filter.	6.1.3.1
<b>higain</b>		Set preamp to high gain.	
<b>histeq</b>		Set the display intensity mapping to histogram equalization.	
<b>homemot</b> [ <i>i j k ...</i> ]	integer	Home the specified motors (or all if no arguments are given).	
<b>impupil</b>		Insert the pupil imaging lens.	
<b>j</b>		Insert the J filter.	6.1.3.1
<b>k</b>		Insert the K filter.	6.1.3.1
<b>lamp</b> <i>name</i> [ <b>on/off</b> ]	text	Turn lamp <i>name</i> on or off.	6.1.10.3
<b>lamp</b> <i>name</i> [ <i>y/slit</i> ]	text, float or text	Insert lamp <i>name</i> to either position <i>y</i> (if a floating point argument is provided) or to the position of the slit (as parametrized by the location of the slit mask) if “slit” is given.	6.1.10.1, 6.1.10.2
<b>lamps</b> [ <b>off/stow</b> ]	text	Turn all lamps off and (optionally) stow them out of the beam.	6.1.10.4, 6.1.10.5
<b>lastfile</b>		Shows the last file number.	
<b>lhetemp</b>		Report LHe temperature.	6.1.2.1
<b>lindisp</b> <i>min max</i>	float	Linear stretch display between <i>min</i> and <i>max</i> .	
<b>ln2</b>		Report LN2 temperature.	6.1.2.1
<b>loadstate</b> <i>name</i>	text	Read new configuration from file <i>name</i> and write it to keywords.	3.5.2, 3.5.4
<b>logain</b>		Set preamp to low gain.	
<b>malign</b>		Run the MAlign procedure to focus the secondary, optimize its tilt to remove coma, and restack the primary segments.	6.1.12.3
<b>mark</b>		Stores the current telescope offsets (generally for later use with <b>gomark</b> ).	
<b>mosaic</b> <i>nx ny dx dy</i>	arcsec	Move and take <b>goi</b> frames in <i>nx</i> rows, <i>ny</i> cols, with ( <i>dx,dy</i> ) offsets.	6.1.12.1
<b>mov</b> <i>x1 y1 x2 y2</i>	pixels	Move an object from ( <i>x1,y1</i> ) to ( <i>x2,y2</i> ) on the detector.	6.1.11.5
<b>mskclr</b>		Remove the slit mask from the beam.	6.1.6.3
<b>mxy</b> <i>x y</i>	arcsec	Move the telescope ( <i>x,y</i> ) arcsec in instrument coordinates.	6.1.11.3
<b>n</b> <i>n</i>	arcsec	Move the telescope north <i>n</i> arcsec.	6.1.11.1
<b>nextfile</b> <i>n</i>	integer	Set the next file number to <i>n</i> . If the value is left off, it will set the next file number to one greater than the largest number in the current output directory.	6.1.1.2.2
<b>obj</b> <i>str</i>	text	Set the OBJECT FITS keyword to <i>str</i> .	6.1.15.2
<b>object</b> <i>str</i>	text	Same as <b>obj</b> .	6.1.15.2
<b>observer</b> <i>names</i>	text	Set observer names.	6.1.15.1
<b>odiff</b>		Display the difference between the last two frames.	5.3.2.2
<b>ostat</b>		Show the statistics from the last picture.	5.3.2.4, 5.3.2.5
<b>pause</b>		Pause to await user action before continuing with script.	
<b>pdiff</b> <i>m n</i>		Display the difference between frame <i>m</i> and frame <i>n</i> .	5.3.2.2

<b>preslit</b>		Set the preslit to a position which will baffle a slit when inserted, or baffle the imaging mode when the slits are cleared. This command will be done from within the “slt*” and “sltclr” commands.	6.1.8.1
<b>psltclr</b>		Clear the preslits.	6.1.8.2
<b>pstat <i>n</i></b>	integer	Show statistics from file <i>n</i> .	5.3.2.4, 5.3.2.5
<b>pupil <i>text</i></b>	text	Insert pupil mask “ <i>text</i> ” where <i>text</i> is one of a list of pupil mask names: circ, insc, hex1, hex2, ..., square, etc. Pupil mask complement TBD.	6.1.7.1, 6.1.7.2, 6.1.7.3, 6.1.7.4
<b>px <i>x</i></b>	float	Move the telescope a distance <i>x</i> in detector focal plane coordinates.	6.1.11.4
<b>pxy <i>x y</i></b>	float	Move the telescope a distance ( <i>x,y</i> ) in detector focal plane coordinates.	6.1.11.4
<b>py <i>y</i></b>	float	Move the telescope a distance <i>y</i> in detector focal plane coordinates.	6.1.11.4
<b>readmode <i>text</i></b>	text	Set the readout mode (Fowler, pair reset, etc.).	6.1.1.2.3, 3.3.2
<b>runname <i>text</i></b>	text	Set the data directory to <i>text</i> .	6.1.13.1
<b>s <i>s</i></b>	arcsec	Move telescope <i>s</i> arcsec south.	6.1.11.1
<b>s5</b>		Move object along the slit to 5 positions, taking frames each time.	6.1.12.1
<b>savestate <i>name</i></b>	text	Save configuration keywords to file <i>name</i> .	3.5.2, 3.5.4
<b>scent <i>x y</i></b>		Move object at ( <i>x,y</i> ) to (180,128), the nominal slit center (as read from the mask position).	6.1.11.19
<b>sdiff</b>		Display the difference between the last two frames. (Has the opposite sign from <b>odiff</b> ).	5.3.2.2
<b>sgr <i>n [filt]</i></b>	integer, text	Select grism number <i>n</i> in the grism slide and optionally insert blocking filter <i>filt</i> .	6.1.5.1, 6.1.5.4
<b>showtemp</b>		Shows the coldhead temperatures.	
<b>skey</b>		Show all instrument keywords.	
<b>slt1 <i>y</i></b>	float	Insert slit 1 to position <i>y</i> on the detector. There will be other similar commands for other slits, TBD when the slit complement is finalized.	6.1.6.1
<b>sltclr</b>		Clear all slits and mask out of the beam.	6.1.6.3, 6.1.9.2
<b>sltmclr</b>		Clear the mask for the slits.	6.1.6.3
<b>sltmov <i>y</i></b>	arcsec	Move the object along the slit by <i>y</i> arcsec.	6.1.11.20
<b>sltsclr</b>		Clear all slits out of the beam.	6.1.6.3
<b>snapi [<i>n</i>]</b>	integer	Take <i>n</i> pairs of target and sky exposures. (The current nod parameters define the sky position. If <i>n</i> is not specified, a single pair is taken.	6.1.12.2
<b>snapiv [<i>n</i>]</b>	integer	Take <i>n</i> pairs of target and sky exposures using <b>snapi</b> and display the difference.	6.1.12.2
<b>sp2 <i>n</i></b>	integer	Take <i>n</i> exposures, moving 2 positions along slit separated by 12.8 arcsecs.	6.1.12.1
<b>sp55 <i>n</i></b>	integer	Take <i>n</i> exposures, moving between 5 positions along slit separated by 5 arcsecs.	6.1.12.1
<b>sp56 <i>n</i></b>	integer	Take <i>n</i> exposures, moving between 5 positions along slit separated by 6 arcsecs.	6.1.12.1
<b>sp74 <i>n</i></b>	integer	Take <i>n</i> exposures, moving between 7 positions along slit separated by 4 arcsecs.	6.1.12.1
<b>subc <i>nx [ny]</i></b>	integer	Select a centered <i>nx</i> x <i>nx</i> or <i>nx</i> x <i>ny</i> subarray.	6.1.1.2.4, 3.3.2
<b>thinxy <i>dx dy n</i></b>	arcsec, integer	Take <i>5n</i> exposures, using a box with ( <i>dx,dy</i> ) steps, plus the central position, displaying each.	6.1.12.1

<b>tint</b> $[t]$	seconds	Set integration time to $t$ seconds. If $t$ is not specified, the current integration time is reported.	6.1.1.2.4
<b>tosky</b>		Offset the telescope by the nod parameters.	
<b>w</b> $w$	arcsec	Move telescope $w$ arcsec west.	6.1.11.1
<b>wd</b> $n$	integer	Write scratch buffer $n$ to the data directory as a new image.	6.1.13.3
<b>x</b> $x$	arcsec	Move telescope $x$ arcsec in instrument coordinates.	6.1.11.3
<b>xy</b> $x y$	arcsec	Move telescope $(x,y)$ arcsec in instrument coordinates.	6.1.11.3
<b>y</b> $y$	arcsec	Move telescope $y$ arcsec in instrument coordinates.	6.1.11.3
<b>zero</b>		Return the telescope to its base position (i.e. move to zero offset position).	
<b>Ctrl-C</b>		Abort a command, return to system prompt.	6.1.1.3, 6.1.1.4

### 3.1.2 Command Script Examples

It may prove useful to dissect a sample Unix shell script. We will choose an existing NIRC dither pattern, `bxy9`, as an example. This script will be used with NIRC2 as well, with relatively minor changes (e.g., using the NIRC2 keyword library rather than the NIRC1 library).

The script itself follows:

```
#!/bin/csh -f
#
#bxy9 x y moves the telescope in a 9-position box pattern, using
#offsets of x arcsec and y arcsec in detector coordinates.
#
#Pattern is:      2 8 4
#                6 1 7
#                3 9 5
#
if ({#argv} == 2) then
    set x = $1
    set $y = $2

else if ({#argv} == 1) then
    set x = $1
    set $y = 1

else
    echo "Usage: bxy9 x y"
    exit

endif

#Set matrix of offset values.

set xoff = ( 1 0 -2 0 2 -2 1 0 0)
set yoff = (-1 2 -2 2 -1 0 -1 2 -1)
foreach pos (1 2 3 4 5 6 7 8 9)
    echo ''''
    echo Position $pos of 9...
    goi
    mxy 'echo "$xoff[$pos]*$x"|bc -Im' 'echo "$yoff[$pos]*$y"|bc -Im'
    odiff>&/dev/null &
end
```

The first line tells Unix that we will use the csh shell, and that it should not run the .cshrc file. The following nine lines are comments, as denoted by the “#” symbol in the first column. This is used in the “help” command; if a user types “help bxy9”, the command will print every line in the script which begins with a “#” sign. This provides a simple help facility which uses the already-existing documentation inside each command script, with no extra effort. Of course, more detailed descriptions are provided elsewhere, such as the user’s manual.

Error checking can be done in one of several ways; this script simply checks to make sure that either one or two arguments have been passed. If the number of arguments is 0 or 3 or more, an error message is printed. If only one argument is passed, the dither pattern is assumed to have step sizes equal in both x- and y-directions. If two arguments are passed, the first is taken as the x-step and the second as the y-step.

Note that this ability to easily allow different numbers of arguments is a powerful advantage over some other tools, such as the C programming language itself.

Next, two arrays are loaded with the offset directions and magnitudes. These are multipliers of the x- and y-sizes passed as arguments.

The foreach loop increments the counter “pos.” For each position, a message is printed (to let the observer know where they are in the script) and an image taken using the script “goi.” “Goi” is yet another Unix shell script that takes a single image. After the image has been taken, the telescope is moved using a third Unix script, “mxy,” which takes as arguments x- and y-offsets in arcseconds. Note that some math must be performed within the mxy call, to multiply the offset values by the appropriate, indexed multipliers.

The “odiff” command sends to the display the difference between the last and previous-to-last image, providing a pair subtraction that allows the observer to see what the images are providing. A twist here is that the display command is spawned off as a separate process, so that the image-taking can proceed and not wait until the display has been refreshed. This shaves a small number of seconds off the script’s runtime and decreases the observing overhead. To avoid confusing the observer with the output of odiff, that output is sent to /dev/null, which effectively removes it from the standard output.

Once the first image is taken and displayed, the loop increments “pos” and proceeds to take the next image.

This is a facility script, and in some sense has more sophisticated processing than the observer’s own personal scripts need. Given the facility scripts, it is even easier for observers to string together a series of commands to produce their own custom script. One example comes from an actual observer’s script used with NIRC:

```
#!/bin/csh -f
#
#byr takes the standard byr survey image sequence.
#
#Nod is by default 40". Image sequence is as follows:
#
#Ks: on-off-on
#CH4: on-off-on
```

```

#Ks: on-off-on
#CH4: on-off-on
#Ks: on-off-on
CH4: on-off-on
#
#Standard integration times: Ks: tint 0.5; coadd 120
#                               CH4: tint 1.0; coadd 60
#
#Argument is object name
#
if ({#argv}==1)then
    set name = $1

    object `echo "$name" `
    node 60
    nodn 40
    foreach pos (1 2 3)
        tint 0.5
        coadd 120
        ks
        snapi
        goi
        tint 1.0
        coadd 60
        ch4
        snapi
        goi
        echo Set number $pos has finished for $name.
    end
    echo "Usage: byr objname"
endif

```

After checking for a single argument, this script sets the object name to that argument, sets parameters to nod the telescope E (by 60 arcseconds) and N (by 40 arcseconds), then loops through a series of commands three times.

The loop starts by setting the integration time to 0.5 sec, and the number of coadded frames to 120. The K-short filter is then inserted using the “ks” command. The “snapi” command takes a pair of exposures, one at the target position and one offset by the nod distance set near the start of the script. After returning to the target, a single exposure is taken (using “goi”). The integration time is then set to 1 second and the coadds to 60 for the second filter, which is the CH4 filter. Again a trio of exposures is taken. A status message is printed and after the loop is finished the terminal bell is rung three times (“bells 3”) to alert the observer that the script has completed.

Note in particular that this script requires relatively little Unix expertise. In fact, if the check for correct number of arguments were removed, the only Unix knowledge necessary would be how to refer to a passed argument (“object \$1” could have been used in this script), how to create a loop (foreach), and how to print a status message (the “echo” command).

Some observers, of course, have more Unix expertise, and have written quite sophisticated scripts for their own use.

### 3.1.3 Graphical User Interface

Like its predecessor, NIRC1, NIRC2 will make minimal use of GUI technology. Other than the Quick Look display tool (§3.6.2), the only GUI will be a read-only status screen of selected keywords. As in NIRC1 (Figure 2), the display will be implemented via the generic keyword display tool “xshow.” Figure 3 depicts a similar xshow configured for the relevant NIRC2 keywords.

Although we do not include extensive use of GUI technology in this design, we note that a GUI upgrade path exists thanks to the keyword API that underlies the CLI. Within CARA we have developed methods for interfacing each of the popular GUI construction packages (tcl/tk, IDL, java, C/motif, and DataViews) to keyword servers.

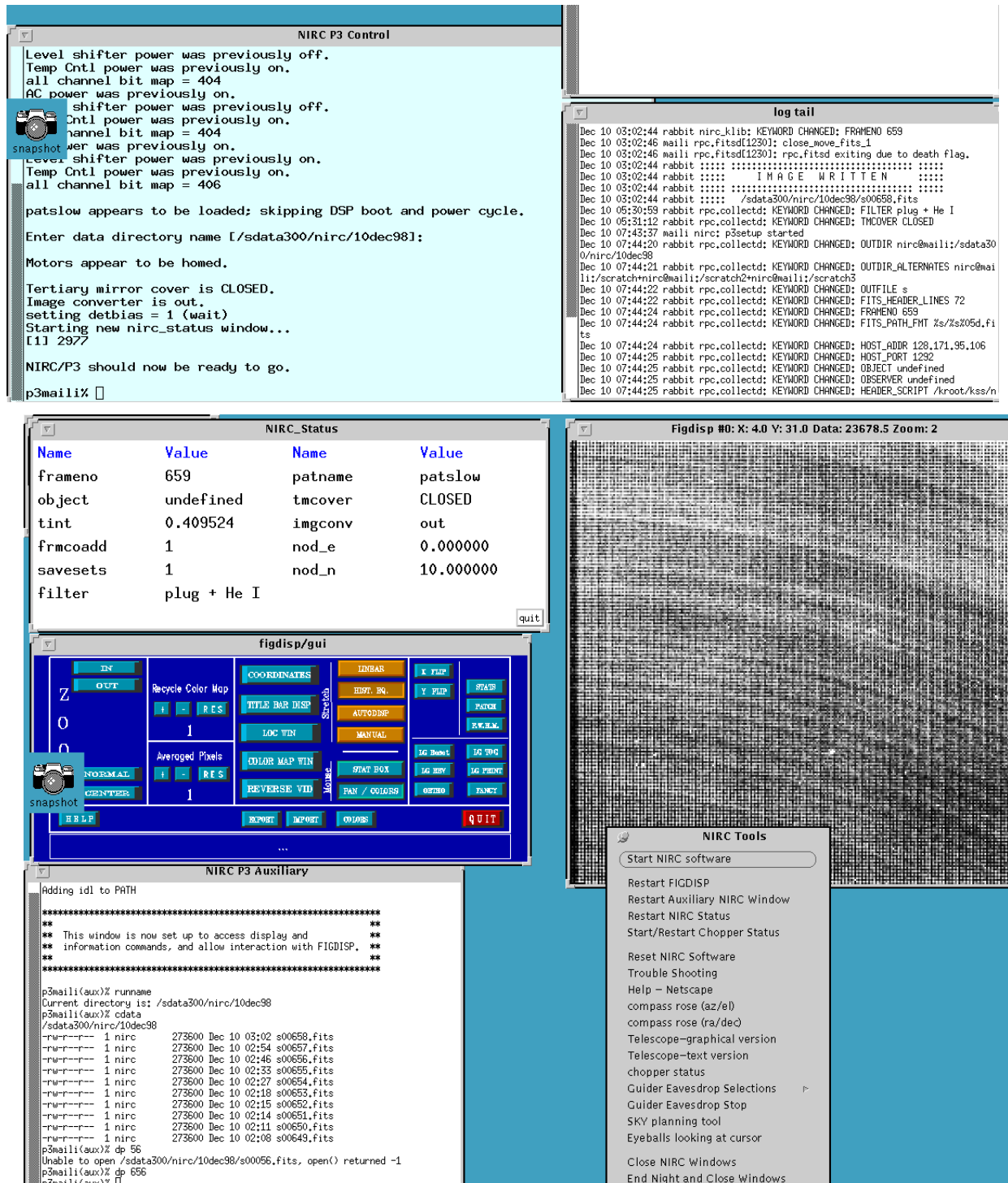


FIGURE 2. Screen Dump of a NIRC1 Session.

NIRC2-Status			
Name	Value	Name	Value
frameno	659	patname	patslow
object	undefined	tmcover	CLOSED
tint	0.409524	imgconv	out
frmcoadd	1	nod_e	0.000000
savesets	1	nod_n	10.000000
filter	plug + He I	grism	GR120

FIGURE 3. Status Display

### 3.2 Mechanism Keyword Library

A shareable mechanism keyword library (libnirc2m\_keyword.so.0) will be written in the C programming language to provide keyword functions for the control and status of the NIRC2 mechanisms described in Table 2.

In addition to the motor-controlled mechanisms, the NIRC2 instrument has several other associated devices. These include the temperature controller, coldhead, and various thermistors.

The proposed keywords for these devices are enumerated in Table 6.

An additional category of observing keywords will be user-defined keywords which will be used as user variables to extend the NIRC2 functionality. The intent of these keywords is to provide persistence between keyword library invocations. We use this technique in the P3 system (§1.1.2) and it has proven valuable in extending NIRC functionality. For example, Gary Chanan's group at U.C. Irvine has recently written a set of commands which phase the Keck I primary mirror segments using out-of-focus NIRC images. The variables used by their procedures are kept as NIRC keywords, allowing persistence between different scripts as well as between different users. In the latter sense, user keywords are more powerful than Unix environment variables, which are generally set only for the current process and its daughter processes. The P3 implementation allows, for example, someone in Irvine to see exactly what parameters are in use by the observer in Hawai'i.

The user keywords will be implemented by storing  $\langle name, value \rangle$  pairs within the motor or i/o daemon, such that writing a keyword of the given *name* will cause the specified *value* to be stored in a daemon and reading the keyword will return the *value*.

The values of user keywords will be restricted to strings, which may be numeric. The names of the keywords must be alphanumeric (plus underscores) and must not begin with numbers nor any of the prefixes from Table 3.

**TABLE 2. Optical bench mechanisms.**

<b>Mechanism</b>	<b>Description</b>
camera assembly	single-axis motor-driven linear slide that moves horizontally in a continuous motion between two end stops.
grism slide	single-axis motor-driven linear slide that moves the grisms and pupil reimaging lens assembly horizontally in a continuous motion between two end stops.
shutter	single-axis motor-driven shutter which rotates between two stops. The shutter is positioned between the inner filter wheel and the grism slide.
filter wheels	two single-axis motor-driven wheels each with 18 discrete positions for filters, grisms, or polarizing prisms.
pupil mask	single-axis motor-driven wheel with 6 discrete positions for masks. The wheel is positioned between the outer filter wheel and the collimator. The rotation of the wheel, coupled with the pupil rotation drive determines the angle of the pupil masks (Figure 4).
pupil rotation drive	single-axis motor-driven wheel which is driven in a 'tracking' mode so as to match the telescope pupil orientation (Figure 4).
slits	two single-axis motor-driven linear slides (slit slide and slit mask), moved vertically in a continuous motion between two end stops.
preslits	two single-axis motor-driven linear slides (lower and upper) that move the assemblies vertically to position apertures and edges so as to baffle the beam.

The keyword library will encapsulate the knowledge of the instrument's mechanisms, including which physical devices are associated with which mechanisms, through the use of a set of configuration files (section 3.2.6).

The keyword library interface will be the expected/standard functions of `keyword_open`, `keyword_ioctl`, `keyword_read`, `keyword_write`, `keyword_close`, `keyword_event`, and `keyword_respond`, which will be used by the Keck tasking library programs (`show`, `xshow`, and `modify`; Appendix D).

The keyword library will communicate with motor and I/O daemons via rpc calls.

The set of available keywords will be provided by a configuration file that will be read and parsed when the keyword library is invoked. If a keyword is not in the config file then the keyword will not exist, thus, it will be possible to tailor the set of available keywords to groups of users.

None of the keywords will be case sensitive in regard to the 'show' and 'modify' programs.

The mechanism keywords will have three broad categories: observing keywords that will typically be used by the users; maintenance keywords that users will never use but may be used by instrument specialists to troubleshoot and/or formulate workarounds; engineering keywords that may be used for troubleshooting, integrating, and configuring devices. The categories are described in subsequent subsections.

### 3.2.1 Observing keywords

The observing keywords will encompass those keywords which a user would typically use (via scripts, §3.1) to manipulate the NIRC2 mechanisms. These keywords will be in terms of the mechanisms, for instance, a keyword to move the outer filter wheel to a discrete position will have an acronym (fwopos) that will distinguish it as such. These keywords will not refer to specific motors.

Each keyword associated with a mechanism that is motor controlled will have a mechanism-specific prefix which will be defined in a configuration file. The prefixes to mechanisms' associations are provided in Table 3.

**TABLE 3. Prefixes to mechanisms' associations.**

Prefix	Mechanism
CAM	camera slide
FWO	outer filter wheel
FWI	inner filter wheel
GRS	grism slide
PRDW	pupil rotation drive
PSL	lower preslit
PSU	upper preslit
SHTR	shutter
PW	pupil wheel
SLTM	slit mask
SLTS	slit slide

With the exception of the pupil rotation drive, all the mechanisms will be moved to discrete positions either rotationally or linearly. Thus, all of them, with the one exception, will have the same set of keyword suffixes as provided, along with their functional descriptions, in Table 4.

Because there are, excluding the pupil drive, 10 prefixes in Table 3 and 10 suffixes in Table 4, these tables define 100 keywords. For example, CAMNAME is the named position of the camera slide, FWIRAW is the raw position of the inner filter wheel, etc.

The pupil rotation drive (PRDW) requires a set of keywords that contains most of the suffixes used by the other mechanisms and a few additional ones. Those keyword suffixes in Table 4 that will not apply to the pupil rotation drive are designated as such in that table. The suffixes unique to the PRDW are provided in Table 5.

The combination of a keyword prefix and suffix creates a unique keyword which specifies the action to be performed on a specific mechanism. The resulting keywords will be enumerated in a configuration file, along with other characteristics (§3.2.6) and are enumerated in Appendix 3.

In addition to the motor-controlled mechanisms, the NIRC2 instrument has several other associated devices. These include the temperature controller, coldheads, and spectral lamps.

The proposed keywords for these devices are enumerated in Table 6.

**TABLE 4. Keyword suffixes and their descriptions.**

Suffix	Description
DELTA	These write-only integer keywords will allow one to offset mechanisms' positions. The offset values will be in motor counts. Operationally, one would seldom use these keywords.
EUP	These 'engineering unit position' keywords will be read/write floating point keywords. The positions will be in engineering units, which for wheel devices is degrees (modulus 360) and for translational devices (slides) will be millimeters (note that the scaling to motor counts is specified in a configuration file discussed in §3.2.6).  Reading these keywords will return the current absolute positions of mechanisms, in the relevant units. Writing these keywords will move mechanisms to absolute positions.
HOME	These read/write keywords will be boolean. Setting these keywords to true will cause mechanisms to perform their homing sequences if the associated motors are not moving (setting to false has no effect).  Reading these keywords will indicate if the mechanisms' homing sequences have completed since the last reset or powerup of their associated motor controllers.
NAME	These will be read/write string keywords. Setting these keywords will cause mechanisms to move to the specified 'named' positions (designated in a mechanism-specific config file, which is the domain of the instrument specialists).  Reading these keywords will return mechanisms' current 'named' positions or "UNKNOWN" if an associated motor is not at a known discrete position. A 'NAME' keyword is meaningless with respect to the pupil rotation drive (PRDW prefix).

**TABLE 4. Keyword suffixes and their descriptions.**

<b>Suffix</b>	<b>Description</b>
POS	<p>These will be read/write integer keywords whose values will be numbers representing mechanisms' discrete positions, as defined in mechanism-specific config files, which is the domain of the instrument specialists.</p> <p>Writing these keywords will cause mechanisms to move to the specified discrete positions.</p> <p>Reading these keywords will return the position numbers associated with mechanisms' current motor positions, or -1 if a motor position is not sufficiently close (defined in a config file) to a known discrete position. A POS keyword is meaningless with respect to the pupil rotation drive (PRDW prefix).</p>
RAW	<p>These will be read/write integer keywords whose values will be in motor counts. Reading these keywords will return the absolute position of mechanisms' motors. Writing these keywords will move mechanisms to absolute motor positions. Operationally, one would seldom use RAW keywords, as POS, EUP, or NAME are preferred.</p>
DEST	<p>These integer keywords will provide the absolute motor counts to which a mechanisms motor was last demanded. While a motor is moving a DEST keyword will be the destination and the associated 'RAW' keyword, will be periodically updated with the current motor position.</p>
IDLE	<p>These boolean keywords will provide the idle status for mechanisms' motors, where 1 implies idle.</p>
STAT	<p>These string keywords will provide the current state of mechanisms' motors. The strings are MOVING, IDLE, HOMING, RESETTING, KILLING, STOPPING, RESET, POWER_OFF, and UNKNOWN (prior to the first motor commands).</p>
TRGT	<p>These string keywords will provide mechanisms' destinations in 'named' positions, which will be mechanism-specific and designated in a mechanism-specific config file, which is the domain of the instrument specialists. While a motor is moving a TRGT keyword will be the destination and the associated 'NAME' keyword, will be periodically updated with the current named position.</p> <p>A "TRGT" keyword is meaningless with respect to the pupil rotation drive (PRDW prefix).</p>

**TABLE 5. Pupil rotation drive unique keywords.**

<b>Keyword</b>	<b>Description</b>
PRDWVEL	This will be a read/write floating point keyword associated with the pupil rotation drives velocity.  Writing this keyword will set the PRDW velocity to that specified.  Reading this keyword will return the current velocity of the PRDW.

**TABLE 6. Thermal and spectral lamp keywords.**

<b>Keyword</b>	<b>Description</b>
CHEAD_REMOTE	allows switching the CTI coldheads between manual and remote control
CHEAD1PWR	power control of the single stage coldhead
CHEAD2PWR	power control of the double stage coldhead
CHEAD1SPD	speed control of the single state coldhead
CHEAD2SPD	speed control of the single state coldhead
DETSTPT	detector temperature set point
BENCHSTPT	optical bench temperature set point
DETBLCKT	temperature diode one
PRESLIT_T	temperature diode two
CAMERA_T	temperature diode three
CHEAD_SHIELD_T	temperature diode four
SHIELD_T	temperature diode five
CHEAD_BENCH_T	temperature diode six
CHEAD_DETECTOR_T	temperature diode seven
DETHEADT	temperature diode eight
GETTER_T	temperature diode nine
SLIT_T	temperature diode ten
LAMPPWR	global spectral lamp on/off control
ARGONPW	argon spectral lamp on/off control
KRYPTONPW	krypton spectral lamp on/off control
NEONPW	neon spectral lamp on/off control
XENONPW	xenon spectral lamp on/off control

### 3.2.2 Maintenance keywords

The maintenance keywords will be comprised of mechanism keywords that would not typically be used for observing, but may be used by an instrument specialist to troubleshoot problems. These keywords will be similar to the observing keywords defined in §3.2.1 in that they are at the mechanism level, as opposed to the motor level, and will consist of a prefix and a suffix.

The proposed maintenance keyword prefixes are enumerated in Table 3 and the suffixes are enumerated in Table 7.

**TABLE 7. Maintenance keyword suffixes.**

Suffix	Description
BRAKES	These boolean keywords will allow one to control the released/set state of the brakes connected to mechanisms' motors. The brakes will normally be set such that they must be energized (set keywords to false) in order to be released. The brakes will be automatically released at the beginning of motor motion and set when motion completes.  Note that when brakes are released, current flows and heat is generated.
ENABLE	These boolean keywords will allow control of the on/off state of mechanisms' motors. When the motors are enabled (set keywords to true), current flows and heat is generated.
INFO	When these boolean keywords are written to, the mechanisms' position tables will be displayed (info loaded from mechanism-specific config files). The info will include: associated motor number; motor counts per wheel rev (wheels) or per millimeter (sliders); number of discrete position; and for each discrete position the position number, angle (wheels) or distance (sliders), hysteresis, and name.
KILL	Writing true to these boolean keywords will cause mechanisms' motors to abruptly stop moving. Writing false has no effect.
RESET	Writing true to these boolean keywords will cause mechanisms' motors to perform power on resets if the motors are not moving. Note that homing is recommended after resets. Writing false has no effect.
SIM	Writing true to these boolean keywords will cause the keyword library to place mechanisms into simulation mode. Please refer to the programmer's guide for a discussion on simulation.
STOP	Writing true to these boolean keywords will cause mechanisms' motors to decelerate until stopped. Writing false has no effect.

### 3.2.3 Engineering keywords

The engineering keywords will be comprised of low-level motor-specific keywords. These keywords are divided into general purpose and motor-specific as discussed in the following subsections.

### 3.2.4 General purpose motor keywords

Some of the engineering keywords will essentially trigger specific motor actions, such as homing, or resetting. These keywords will all be write-only and take a motor number as an argument, consequently they will all be integer keywords.

If one specifies a motor number of 0 then the action will be applied to all configured motors.

These keywords will all have prefixes of 'mtr', and suffixes provided in Table 8.

Note that the only keyword in this category which will cause motor motion is the 'home' command. When the home command is issued the brakes and servos will be automatically controlled (this is in contrast to the motion keywords provided by the motor-specific keywords discussed in the next section).

#### 3.2.4.1 Motor-specific keywords

Some engineering keywords will be motor-specific and will have a motor number embedded in their spelling. Thus, the keywords consist of a prefix 'mtr', followed by a motor number, and a suffix provided in Table 9.

These keywords will not be restricted to trigger type actions and provide for motor motion commands. However, the setting/releasing of brakes and enabling/disabling of servos will not be done automatically, such that the user will need to modify the state of several keywords to cause motion to occur.

**TABLE 8. General purpose motor keywords.**

Suffix	Description
BRAKESON	causes the specified motor's brakes to be set
BRAKESOFF	causes the specified motor's brakes to be released
DISABLE	causes the specified motor's servo to be turned off
ENABLE	causes the specified motor's servo to be turned on
HOME	causes the specified motor to perform its homing sequence
INFO	displays the motor characteristics for the specified motor. The displayed information is that contained in the motor config file (§3.3.2).
ISHOME	will return true if the specified motor has successfully completed its homing sequence since last power-up or reset, false otherwise.
KILL	causes the specified motor to abruptly stop moving
POSN	returns the current position, in motor units, of the specified motor
RESET	causes the specified motor to perform its reset sequence
STOP	causes the specified motor to decelerate to a stop
WAIT	waits for the specified motor to stop moving. This will timeout, where the timeout duration is specified in a configuration file (§3.2.6).

**TABLE 9. Motor-specific keywords.**

<b>Suffix</b>	<b>Description</b>
ASK	allows users to communicate with motor controllers in motor-specific command strings to which the motor controllers will generate replies. The replies will be printed.  The commands strings will not be parsed or manipulated by either the keyword library nor the motor daemon, so the onus will be on the user to ensure the commands are syntactically correct for the motor controllers.
HOME	reading will return 1 if the associated motor has been homed since its last powerup and/or reset, and 0 otherwise.  writing will cause the associated motor to commence its homing sequence.
IDLE	will return one if the relevant motor is in an idle state, a 0 otherwise.
INFO	will cause a motor's characteristics to be printed
RAW	reading will return a motor's current motor position. writing will cause a motor to be moved to the specified absolute motor position.
DELTA	write-only keywords which will cause the relevant motors to make relative moves of the specified amounts.
TELL	allows users to communicate with motor controllers in motor-specific command strings. No motor controller responses are expected and any that are generated will be discarded.  The command strings will not be parsed or manipulated by neither the keyword library nor the motor daemon, so the onus will be on the user to ensure the commands are syntactically correct for the motor controllers.

### 3.2.5 Moving motor behavior

Many of the mechanism keywords will cause motor motion. The behavior of the keyword library in response to a command which causes motor motion will be determined by whether or not a *nowait* option is specified (appendix D). The default will be to *wait*.

#### 3.2.5.1 Motion keywords' behavior

The default wait behavior will affect the motion keywords (POS, NAME, EUP, HOME, DELTA, and RAW) such that a wait will occur if a motor is not 'idle' prior to issuing the new move, and a wait for a return to the idle state will occur after the move commences. The waits will occur with timeouts that are settable in a config file and will default to 180 seconds.

If the *nowait* flag is set then the motion keywords will generate an error if the motor is not idle prior to issuing the move command to the motor, and command completion will occur immediately after it is issued.

### 3.2.5.2 Halting keywords' behavior

The default wait behavior will affect the STOP and KILL keywords such that command completion will not occur until the relevant motor is 'idle'. In the case of the nowait flag being set, command completion will occur immediately after issuing the relevant motor command.

### 3.2.5.3 Status keywords' behavior

Irrespective of the nowait option, the status or feedback keywords will all be updated at the beginning of a motor move (STAT to 'moving', DEST to the destination position in motor units, and TRGT to the name of the destination position), and STAT will be updated to 'idle' when the move terminates.

If the nowait option is specified then STAT will be set to 'moving', POS will be set to -1, and NAME will be set to 'undefined', when the move commences.

If the nowait option is not specified then STAT will be set to 'moving' and POS, NAME, and RAW will be updated at 1hz until motion terminates or the move timeouts. Upon successful move completion STAT will be set to 'idle'.

## 3.2.6 Configuration files

The keyword library will be driven by a set of configuration files which will be read when the keyword\_open function is invoked from the KTL software layer. These configuration files will specify which keywords are valid, which mechanisms exist, the characteristics of those mechanisms and how they relate to the motor and i/o devices. This permits maximum flexibility with relative ease. However, it is expected that modifications of the configuration files will rarely be done and only be done by an instrument specialist.

### 3.2.6.1 Keyword config file

A configuration file will exist with the purpose of specifying the keywords and their characteristics.

The keyword library will attempt to locate the configuration file via a unix environment variable. If the env variable does not exist then the library will search in /kroot/rel/default/lib and the local directory for the config file. This mechanism will provide for alternative configuration files during testing and integration, and for restricting sets of keywords to specific groups of users.

The keywords will adhere to the naming restrictions discussed in §3.2.

The characteristics to be specified are enumerated in Table 10.

**TABLE 10. Fields of the keyword configuration files.**

<b>Characteristic</b>	<b>Description</b>
Keyword name	The string used by users to refer to a keyword. All keywords associated with motor-controlled mechanisms must adhere to the prefix and suffix restrictions discussed in §3.2. Keywords associated with other i/o devices may include any alphanumeric character as well as underscores, however, the name must not begin with a number. Although keyword names are restricted to 32 characters, those keywords which are to be included in FITS headers must not be greater than 8 characters in length.
Description	The description is a string that should be somewhat descriptive of the keywords function. The description is that which is displayed when 'show' is used to display the list of keywords. A description must be no longer than 72 characters.
Units	The units strings are a descriptive field, such as “mm” or “arcsec,” that will be reported in a <i>show</i> or <i>xshow</i> command. The field must be no longer than 32 characters.
Data type	The data type field is necessary and specifies the type of data expected and/or returned by the keyword library in response to write/read calls.  A keyword may be of type float, integer, boolean, string, double, or enumerated. In most cases the keyword library will convert to/from the specified data type to its internal data type as necessary. Some of the keywords are permitted to be only string. For instance FWONAME (outer filter wheel names position keyword) must be string. It is hoped that the restricted keywords can be intuitively derived.
Class	The keyword class is essential as it defines the mechanism or device type to which the keyword is associated and, therefore, the processing performed by the keyword library.  The allowed mechanism classes are: wheel, slider, rotator, and motor.  The other device classes are pertinent to the analog and digital i/o signals: DIN, DOUT, AOUT, DAC, AIN, or ADC, along with the device number and signal number.
Precision	For float datatypes, specifies the degree of precision to be used when displaying the value in <i>show</i> or <i>xshow</i> .
I/O flag	This field specifies whether the keyword is readable, writeable, or both.

### 3.2.6.2 Mechanism config file

A configuration file will exist with the purpose of: relating keyword prefixes to mechanisms type; identifying the related motor number; specifying conversion factors from engineering units to motor units; and identifying any additional files containing mechanism-specific characteristics.

The keyword library will attempt to locate the configuration file via a unix environment variable. If the env variable does not exist then the library will search in /kroot/rel/default/lib and the local directory for the config file.

The mechanism configuration file must contain the information specified in Table 11.

Each of the wheel, slider, and rotator mechanisms has characteristics which will be supplied by additional configuration files. Thus, there will be a config file for each wheel mechanism (FWO, and FWI), slider mechanism (CAM, GRS, PSL, PSU, PW, SHTR, SLTM, and SLTS) and rotator (PRDW).

Associated with each mechanism-specific config file will be an environment variable which must be specified in the mechanism config file, as indicated in Table 11. If an env var is defined then it must specify the path and filename of the relevant config file. If an env var is not defined then the keyword library will search /kroot/rel/default/lib and the local directory for the config file specified in the mechanism config file, as indicated in Table 11.

The environment variables will be of the form:

- NIRC2\_FWO\_CONFIG
- NIRC2\_FWI\_CONFIG
- NIRC2\_CAM\_CONFIG

The wheel and slider mechanism configuration files will each consist of a table. Each line in the table will specify the position name, position number, motor units for the position, and an accuracy value which will be used to determine if the mechanism is 'in position'.

**TABLE 11. Fields of the mechanism config file.**

Characteristic	Description
Mechanism type	Used by the keyword library to determine how to process keywords associated with a mechanism, that is, the type (wheel, slider, or rotator) determines which keyword library functions are pertinent to the mechanism.
Keyword prefix	String of up to four alpha-chars that is prepended to all keywords associated with the mechanism in question. The prefixes are enumerated in Table 3.
Major motor number	Instrument-wide motor number of the motor which drives the mechanism in question. This number is used by the keyword library in rpc calls to the motor daemon, thus the number must correlate with a motor's specification in the motor daemon's config file (§3.3.2).
Conversion factor	Specifies how to convert to/from motor units.
Number of positions	Relevant to those mechanisms which will be moved to discrete positions, such as the filter wheels. The value will be used, by the keyword library, for defining internal tables and for range checking.

**TABLE 11. Fields of the mechanism config file.**

<b>Characteristic</b>	<b>Description</b>
Environment variables	Must exist for those mechanisms which have additional characteristics defined in other config files as discussed above.
Config filename	Name of the file containing the mechanism details as discussed above.

### 3.2.7 Logging

Keyword library logging will be performed by syslog calls embedded throughout the keyword library functions. All errors, informational messages, and debug messages will use the Unix syslog system as discussed in §4.2.

### 3.2.8 Keyword-changed events/notifications

This section describes the design for implementing a mechanism by which displayed keywords (in a GUI or xshow) will be updated when their respective values change. The proposed design is taken from NIRC1 and LWS.

Whenever a keyword value is modified a message will be logged into a file. A polling loop, within the keyword library, will notice the updates to the file, read the changes and notify the client (xshow or GUI). The polling loop would, therefore, occur only in those cases when the keyword library is used within the context of a task (xshow or GUI). Most keyword transactions will be done by show and modify, causing simple process and exit use of the keyword library, so the polling will be pertinent in only a few cases.

This simple design eliminates the need to maintain lists of connections to all clients and GUIs, as each library invocation only serves one master, and provides the added benefit that a log file will exist containing all keyword transactions. The method has been tested in the NIRC1 P3 system (section 2.2) and has proven to be a simple and reliable technique for low frequency event notification.

### 3.2.9 Simulation

Two levels of simulation will be provided, one at the mechanism level and one at the motor level.

#### 3.2.9.1 Mechanism simulation

A SIM keyword will be supported for each mechanism (e.g., FWOSIM, FWISIM, CAMSIM, ...) the setting of which will affect motion keywords (NAME, POS, EUP, RAW, and DELTA). If a simulate keyword is set then the writing of a motion keyword will cause the associated position status keywords (TRGT, DEST, and STAT) to be updated, but the motion command will not be issued to the motor daemon, that is, the mechanism will not move.

The SIM keywords will exist for testing/debugging/troubleshooting and will not provide added value to observing.

#### 3.2.9.2 Motor simulation

More extensive simulation will be provided, at the motor level, within the keyword library. This simulation will be controlled by a set of Unix environment variables.

The simulation, when enabled, will affect all motor-controlled mechanisms.

Two levels of motor simulation will exist. One will be controlled by the environment variable `NIRC2_MOTOR_DISCONNECTED` and the other by `NIRC2_MOTOR_SIM`, both of which one will be able to simultaneously enable.

Setting `NIRC2_MOTOR_DISCONNECTED` (e.g., 'setenv NIRC2\_MOTOR\_DISCONNECTED'): will bypass all interaction with the motor daemon (rpc calls will not be made), such that the motor daemon need not exist; waits for homing and motion status will not occur; and waits for move completions will not be performed. This simulation level will essentially test the keyword library itself and will be useful when new keywords are added to the keyword configuration file or the keyword library is extended.

Bypassing the motor daemon will require that status values and return statuses also be simulated. Such is the purpose of the environment variables described in Table 12. The only additional note is that the string "Simulation" will be returned for INFO keywords.

Setting `NIRC2_MOTOR_SIM` (e.g., 'setenv NIRC2\_MOTOR\_SIM 2') will enable motor simulation and will set a timeout value (in the example timeouts would be 2 seconds) which will affect some behavior as described below.

When motor simulation is enabled, commands will be formatted and rpc calls will be made to the motor daemon. However, any daemon responses, and hence, motor controller responses, will be ignored; the user will be able to control the timeout value for non-responding motor controllers; and the keyword library will return the values of the environment variables described in Table 12.

The purpose of this simulation level is to allow one to determine the behavior of the motor daemon, in regards to the formatting of motor controller commands, and yet allow the user to control the response from the keyword library (perhaps to test a new user interface). Of course one may wish to set logging to the appropriate level, so that the log file contains the controller command strings. Note that it will be possible to use this simulation level without having the motor controllers connected and/or powered on as the rpc calls will timeout/fail.

The simulation timeout value will affect commands and statuses in such a way that homing and motion statuses will be set to the value of `NIRC2_MOTOR_SIM_STS`, where 0 means not homed or not idle, and 1 means homed or idle. Normally, a motor will need to be in its idle state for a homing or motion command to be processed, however, during simulation the user will have control over this behavior. Also, waiting for motions to complete will normally occur after a motion command, the user will be able to control the length of motion completion timeout and the completion status, via the environment variables.

Note that this simulation level is overridden by the DISCONNECTED simulation level discussed above.

**TABLE 12. Simulation environment variables.**

Environment var	Description
NIRC2_MOTOR_SIM_POS	response value for reads of position keywords.
NIRC2_MOTOR_SIM_VAL	response value for mechanism and motor keyword reads, with a return statuses of NIRC2_MOTOR_SIM_RET (this should be 1 for success, 0 for minor failures, and -1 for severe faults)
NIRC2_MOTOR_SIM_RET	return value for TELL keywords, and all other keyword writes.
NIRC2_MOTOR_SIM_MSG	response value for ASK keywords, and reads of user keywords
NIRC2_MOTOR_SIM_STS	response value for homing and idle status requests.

### 3.3 Motor daemon

The motor daemon will be a stand-alone task which will provide the interface between the mechanism keyword library and the NIRC2 motors which drive the mechanisms described in Table 2. The motor daemon will be responsible for converting motor demands and requests into syntactically correct motor-specific commands and to perform the communications with the motors.

The normal mechanism for interfacing with the motor daemon will be via the Keck tasking library (KTL). This will include user command line use of show, xshow, and modify (§D.2), as well as scripts and client programs (§3.1). However, it will be possible to create client programs that bypass the keyword layer and directly communicate with the motor daemon via rpc calls. Such will be the case of the support programs such as tellmotor and askmotor (§3.3.3).

For each motor type that is to be supported by the motor daemon, a motor-specific module will exist in an appropriately named subdirectory. As of November 1998, the motor daemon will support Animatics motors and hooks will be added for Compumotor (utilized in generation 1 Keck IR instruments).

During startup, configuration files will read and parsed (§3.3.2).

Irrespective of the underlying motor type, all keyword library interactions with the motor daemon will be via a set of rpc functions which will be as device-independent as possible. In most cases the rpc functions will ultimately call functions in the motor-specific modules. The motor-specific module functions are not addressed here. However, the rpc functions are discussed in the following subsections.

### **3.3.1 Rpc functions**

As previously mentioned, the interface to the motor daemon will be via a set of rpc functions. The intent is that the keyword software and/or client programs need not know any of the underlying motor-specific commands/language but, rather that certain motor functions will be supported. The keyword library will translate the mechanism and motor keywords into the appropriate rpc call to the motor daemon.

Rather than creating a specific rpc function for every possible motor function (or subset thereof), the motor functions are grouped into two main classes, which will be handled by two distinct rpc functions, and six other special purpose rpc functions.

The nature of an rpc interface is such that each rpc function takes a pointer to an argument, which may be a scalar or a structure, and returns a pointer to a scalar or structure. It is the calling program's responsibility to free the memory for the returned item.

The different rpc functions will require different arguments which will undoubtedly require implementing data structures. Herein is described the informational needs of the functions rather than the specific structures needed for implementation.

In the following subsections there are frequent references to 'client'. One client is the keyword library itself, which uses most if not all of the rpc functions. However, the keyword library is not the only client. For instance the motor daemon support programs (§3.3.3) will also be clients.

#### **3.3.1.1 Motor command function**

The motor command function will handle motor commands for which a value is not returned from the associated controller. This function will handle many of the underlying motor operations associated with writing mechanism and motor keywords (POS, NAME, EUP, RAW, DELTA, STOP, KILL, HOME, RESET, ...).

The arguments will be a motor number, a command code which will specify the action to be performed, and a value associated with that action (e.g., a motor destination for a move command).

The response will be the completion status of the command.

#### **3.3.1.2 Motor request function**

The motor command function will handle motor commands for which a value is returned from the associated controller. This function will handle many of the underlying motor status requests associated with reading mechanism and motor keywords (POS, NAME, RAW, STAT, HOME, IDLE, ...).

The arguments will be a motor number and a request code which will specify the action to be performed.

The response will be the value returned from the controller.

#### **3.3.1.3 Motor information function**

The motor information function will handle requests for retrieving motor information obtained from the motor configuration file (§3.3.2). The function will be called in response to processing INFO keywords.

The only argument required is a motor number.

The response will be a string containing a synopsis of the motor information or a string indicating that the motor is either not configured or not responding.

#### **3.3.1.4 Motor existence function**

The motor existence function will be called internally by the keyword library to verify that the arguments to many of the keywords are valid. For instance, if one were to set a motor home keyword to a specific motor number then the existence function would be called to validate the motor number prior to issuing the home command.

Initially, there will be no keywords tied to this function, but a support program may be added.

#### **3.3.1.5 User variable manipulation functions**

A set and a get function will exist for manipulating user keywords (also referred to as user variables as discussed in §3.2.1).

The set function will take <name,value> string pairs.

The get function will take a name string and will return a value string. The caller will be required to free the memory for the value string (standard rpc requirement).

#### **3.3.1.6 Motor controller i/o functions**

A function will exist for sending arbitrary motor controller command strings, and another for sending arbitrary motor controller request strings. The intent is to provide a mechanism by which an expert can use the motor-specific language, the assumption being that the expert is cognizant of motor-specific commands and motor addressing requirements.

In all likelihood the two functions will be “tellmotor” and “askmotor”. Askmotor will wait for a reply from the motor controller, while tellmotor will not. Tellmotor will not cause a failure if a user issues a string to which a reply occurs, the reply will simply be ignored. Askmotor will eventually timeout on a lack of reply which may take several minutes, so the expert should be knowledgeable of the motor commands.

Both functions will require as arguments a motor number in addition to the command/request string.

These functions will handle the ASK and TELL keywords as well as the equivalent support programs (§3.3.3).

### **3.3.2 Configuration files**

When the motor daemon initializes it will read and parse the records contained in a config file. The config file will be identified to the motor daemon as an argument when the daemon task is created. This config file is referred to as the motor config file.

The motor config file has three functions: to identify the communications links; to identify the motors connected to a given link; and to specify the motor characteristics of each of those motors.

More than one communications link may be specified but all motors and their characteristics, for a given link, will be specified before another link is specified. That is, all motor information following a communications link will be assumed to be relevant to that communications link, up to the point that another comms link is specified.

### 3.3.2.1 Communications link specification

The purpose of the comms link spec is to identify the device on which one or more motor controllers are connected.

The motor controllers may be connected to a host (Unix machine) serial port, or to a terminal server port. The syntax of the communications link spec will indicate the type of the connection such that the motor daemon will be able to derive the relevant information.

### 3.3.2.2 Motor specification

For each motor connected to a communications link there will be a motor specification followed by several motor characteristic specifications.

The motor specification will contain the information provided in Table 13.

**TABLE 13. Motor information for specification.**

<b>Motor info</b>	<b>Description</b>
major motor number	the instrument-wide number which must correlate with the number specified for a given mechanism in the mechanism config file (§3.2.6).
minor motor number	the motor address or number to which a specific motor controller, on the comms link in question, will respond. This is typically a motor controller configuration issue. Note there may be multiple motors with the same minor motor number, provided they are on separate communications links, thus the need for the aforementioned major motor number.
intercharacter delay	used when writing to the comms link, if a delay is not specified then characters are output in bursts such that the controllers may not keep up (recommended value is 1 msec between chars).

### 3.3.2.3 Motor characteristics

Each motor will possess a set of characteristics which will be specified in the config file. The exact set required depends upon the motor controller and its programming (NIRC2 motor controllers are currently Animatics SmartMotors).

Each motor characteristic will be on a separate line in the config file.

If a characteristic requires a string of multiple words then that string will be enclosed in double quotes.

How the individual characteristics are handled is a function of the specific motor controller requirements (i.e., a Compumotor motor controller has a different command set from an Animatics motor controller). As mentioned in §3.3, there will be a specific software module for each type of controller, those modules will be responsible for parsing and retaining the pertinent motor characteristics.

All characteristics are defined in Table 14 even though some of them currently are not needed (hooks have been added for other motor controllers such as Compumotor).

**TABLE 14. Motor characteristics.**

<b>Characteristic</b>	<b>Description</b>
resolution	will specify the number of motor counts per motor revolution
low motor limit	will be the smallest permitted demand value in motor counts
high motor limit	will be the largest permitted demand value in motor counts
position error	will specify the maximum allowed position error, in motor counts, when the motor servo is enabled (i.e., when moving). For the Animatics controllers the motor is effectively disabled when the allowed position error is set to 0. Thus, the allowed position error will be set to the value specified in the config file before every move and set to 0 after every move. If the motor enable/disable keywords (§3.2.2 and §3.2.3.1) are used then it will be the allowed position error that will be manipulated
reset	will be a motor command string output to a motor controller to cause the controller to perform its reset sequence.
home	will be a motor command string output to a motor controller to cause the controller to perform its homing sequence.
home status	will be a motor command string output to a motor controller to obtain a status indicating whether or not the motor has been homed.
home status value	will be the value, returned from the home status string (discussed above), which will indicate that homing has been completed.
idle status	will be a motor command string output to a motor controller to obtain a status indicating whether or not the motor is idle (not homing, moving, or executing a trajectory).
idle status value	will be the value, returned from the idle status string (discussed above), which will indicate that a motor is idle.
absolute move	will be a motor command string output to a motor controller to cause the motor to be moved to a specified absolute position.
relative move	will be a motor command string output to a motor controller to cause the motor to be moved relative to its current position.
instrument special	will provide for any special motor subroutine required by a mechanism. This would be a motor command string to cause some non-standard motor action.
acceleration	will provide for setting a motor's acceleration parameter.

**TABLE 14. Motor characteristics.**

<b>Characteristic</b>	<b>Description</b>
velocity	will provide for setting a motor's velocity parameter.
homing velocity	will provide for setting a motor's homing velocity parameter which may differ from its normal motion velocity
backlash	will provide for setting the amount of backlash that a motor must correct for after completing a move.

### 3.3.3 Support programs

Support programs will exist for the purposes of troubleshooting, integration, and motor controller setup. These programs will be intended to be used by the instrument software specialist and/or the instrument specialist and not the user community.

The support programs will bypass the keyword library and interface directly to the motor daemon via the rpc interface. Thus they will all be stand-alone rpc clients which utilize the rpc functions discussed in §3.3.1.

The support programs will be software modules written in the C programming language.

The motor daemon support programs are enumerated in Table 15.

**TABLE 15. Motor daemon support programs.**

<b>Program</b>	<b>Description</b>
askmotor	will allow one to issue a command string, to a specified motor controller, for which a response will be expected and waited for. Multi-word strings will be expected to be enclosed in double quotes. The user will be expected to be knowledgeable of the motor controller language.
tellmotor	will allow one to issue a command string, to a specified motor controller, for which no response is expected and/or ignored. Multi-word strings will be expected to be enclosed in double quotes. The user will be expected to be knowledgeable of the motor controller language.
loadmotor	will allow one to upload or download a specified motor controller program. The user will be expected to be knowledgeable of the motor controller language and its program loading technique. The user will also be expected to be knowledgeable of the communications interface to the motor controller, as loadmotor will require link information to be specified.
motorprop	will allow one to set a specified motor property variable to a specified string or to get the current value of a specified motor property variable. The variables are essentially user keywords/variables (discussed in section 3.2.1) that will be retained in a hash table within the motor daemon.
motorinfo	will allow one to get a display of the motor characteristics of one or all motor controllers.

### 3.3.4 Logging

Motor daemon logging will be performed by syslog calls embedded throughout the motor daemon functions. Syslogging is discussed in §4.2.

The logging level is controlled by the environment variable LOG\_UPTO. This env var must be set within the context in which the motor daemon is launched, in order to affect syslogs from the daemon.

The default logging level will provide for syslogs of errors encountered in the motor daemon. Setting the logging level to 'debug' will provide a logged message for all entry and all successful exits of rpc calls (client interface) and all motor-specific calls (animatics directory), and reading of config files.

An environment variable will be provided to allow enabling debug level logging of the motor command strings and replies, from within the low level send and receive functions.

An environment variable will be provided to enable writing, to the window in which the daemon was launched, of the motor command strings and replies, from within the low level send and receive functions.

### 3.3.5 Simulation

The motor daemon simulation will be minimal as the keyword library simulation is deemed to be more useful.

An environment variable will be provided to enable simulation in the motor daemon to the extent that all i/o to/from the motor controllers will be disabled. If one enables simulation then all formatted messages that would normally be sent to the motor controllers will be written to the window in which the daemon was launched, and all response will be obtained from another environment variable.

The simulation effectively will allow one to test the motor daemon, without any motor controllers, to the extent that the i/o streams can be verified to be correct.

### 3.3.6 Motor controllers

The NIRC2 motor controllers are currently Animatics SmartMotor integrated servo systems. Each controller is a single compact and reliable package consisting of a brushless DC servo motor with brakes, servo controller, encoder, servo amplifier, programmable logic controller, network manager, and 8K EEPROM program memory module.

The controllers are configured for the RS-485 serial protocol which allows for connecting multiple controllers to a single serial link. As there are 11 motors associated with the instrument, the serial protocol eliminates the need for 11 serial links from the instrument to the telescope control room. In fact, we intend to connect the motor controllers to terminal servers thereby eliminating any cabling other than the existing ethernet cable.

The motor daemon design allows for connecting to multiple serial links. This was done so that the pupil rotation drive's motor controller may be connected to a separate serial link, ensuring that the controller will not be hindered by communications to/from the other 10 motor controllers.

### **3.3.6.1 Motor addressing**

The multi-dropping of multiple motor controllers on a given serial link requires that each controller, on a given link, must have a unique address. This is provided for within the Animatics controllers' programming language and is the minor motor number discussed in §3.3.2.2, Table 13).

### **3.3.6.2 Motor setup**

When the motor controllers are received from the manufacturer they are programmed with a default setup, in terms of motor address and serial communications (e.g., baud rate). The default setup precludes a controller from being multi-dropped with other controllers. Procedures will be established for the initial configuration of the motor controllers.

### **3.3.6.3 Motor controller programming**

The Animatics Smartmotor controllers provide a rich command set which will be used to create controller programs that will be stored in the controllers' EEPROM memory modules. The intent is to create a single program for the motor controllers which operate all the instrument mechanisms that are wheel and translational devices (note that the controller addresses will be the only difference). The motor controller which operates pupil rotator will have a different program.

Each of the programs must provide for subroutines to setup control parameters, perform a homing sequence, move to an absolute position, and move to a relative position. Those subroutines which cause motion must enable the motor drive, release brakes, commence motion, wait for motion to complete, set brakes, and disable the motor drive.

The motor drives will be disabled and brakes applied when motion completes to reduce heat generation.

The subroutines for absolute and relative motion must account for mechanical backlash. This will be done by always completing moves in the same direction. The implication being that moves in one direction will always require an additional move of a specific distance. The distance is called the backlash distance and is controlled by a motor characteristic in the motor configuration file (§3.3.2.3, Table 14).

The motor control program for the pupil rotator will differ from the others in that an additional subroutine will exist to provide for motion in velocity mode.

The motor controller programs will reduce the amount of communications between the motor daemon and controllers. Rather than sending long strings of the commands, the motor daemon will typically only need to specify a destination or offset an indication of which subroutine to execute.

### 3.4 I/O Daemon

The I/O daemon will be a stand-alone task which will provide the interface between the mechanism keyword library and the NIRC2 analog and digital i/o devices. The I/O daemon will be responsible for converting lamp, temperature, and coldhead demands and requests into syntactically correct device-specific commands and to perform the communications with the device modules.

Normal interfacing with the I/O daemon will be via the Keck tasking library (KTL). This includes user command line use of show, xshow, and modify, as well as, scripts and client programs. However, it will be possible to create client programs that bypass the keyword layer and directly communicate with the I/O daemon via rpc calls. Such will be the case of the support programs such as “telldevice” and “askdevice” (§3.4.3).

For each I/O module/device type that is to be supported by the I/O daemon, a device-specific module will exist, hopefully in an appropriately named subdirectory. As of November 1998, the I/O daemon will support DGH modules and hooks will be added for Lakeshore and Xycom devices.

During startup, configuration files will be read and parsed (§3.4.2).

Irrespective of the underlying device type on which an i/o signal exists, all keyword library interactions with the I/O daemon will be via a set of rpc functions which will be as device independent as possible. In most cases the rpc functions will ultimately call functions in the device-specific modules. The device-specific module functions are not addressed here. However, the rpc functions are discussed in the following subsections.

#### 3.4.1 Rpc functions

As previously mentioned, the interface to the I/O daemon will be via a set of rpc functions. The intent is that the keyword software and/or client programs need not know any of the underlying device-specific commands/language but, rather that certain I/O functions are supported. The keyword library will translate the analog and digital keywords into the appropriate rpc calls to the I/O daemon.

Rather than creating a specific rpc function for every possible I/O function (or subset thereof), the I/O functions were grouped into analog input, analog output, digital input, digital output, and six other special purpose rpc functions.

The nature of an rpc interface is such that each rpc function takes a pointer to an argument, which may be a scalar or a structure, and returns a pointer to a scalar or structure. It is the calling program's responsibility to free the memory for the returned item.

The different rpc functions will require different arguments which will undoubtedly require implementing data structures. Herein is described the informational needs of the functions rather than the specific structures needed for implementation.

In the following subsections there are references to major device number and minor device number. The distinction is that the minor number is a device-specific number unique to a given communications link, where as the major number is unique across all devices on all communications links. Thus, there could be multiple devices with the same minor number provided that they are all connected to different communication links.

In the following subsections there are frequent references to 'client'. One client is the keyword library itself, which uses most if not all of the rpc functions. However, the keyword library is not the only client. For instance the I/O daemon support programs (§3.4.3) are also clients.

### 3.4.1.1 Analog i/o functions

There will be two rpc functions for reading and writing analog output values.

There will be one rpc function for reading analog input values.

The I/O daemon will hand off processing of rpc calls to functions relevant to the underlying device type (for NIRC2 this is DGH or Lakeshore). Thus, detection of errors, and responses to those errors, will be done on a device-specific basis. For read failures of NIRC2 analog signals, an error will result in a return value of the maximum representable floating value.

**3.4.1.1.1 Writing an analog output.** The function for outputting a value to an analog signal will take as parameters a device number, a channel number, a raw flag, and a floating point output value.

The raw flag will indicate whether the user-supplied value or a lookup table value should be output to the device. If the raw flag is set then the user-supplied value will be scaled and offset (specifications from the configuration file, §3.4.2) then output to the device.

The response will be the completion status where 0 will indicate success.

**3.4.1.1.2 Reading an analog output.** The function for inputting the value of an analog output signal will take as parameters a device number, a channel number, and a raw flag.

The raw flag will indicate whether the device-supplied value or a lookup table value should be returned to the user. If the raw flag is set the input value will be scaled and offset (specifications from the configuration file, §3.4.2) prior to being returned to the user.

If the underlying device does not support readback of outputs then the error value will be returned.`

Some analog devices (DGH modules included) provide a readback of the most recent analog output values, as well as the value of an ADC tied to the DAC. In this case where both are supported, the value of the most recent analog output value will be returned and not that of the ADC. To obtain the value of the ADC feedback one would have to use an analog input read function. This implies there would be two keywords for this signal within the keyword library.

**3.4.1.1.3 Reading an analog input.** The function for inputting the value of an analog input signal will take as parameters a device number, a channel number, and a raw flag.

The raw flag will indicate whether the device-supplied value or a lookup table value should be returned to the user. If the raw flag is set the input value will be scaled and offset (specifications from the configuration file, §3.4.2) prior to being returned to the user.

The response will be the current value of the signal, returned from the module/device, or the error value.

Note that `getadchan_1()` must be used to obtain the value of an ADC channel tied back to a DAC, in those devices which provide such a feature.

### 3.4.1.2 Digital i/o functions

There will be an rpc function for writing digital output values.

There will be an rpc function for reading digital output values.

There will be an rpc function for reading digital input values.

The I/O daemon will hand off processing of rpc calls to functions relevant to the underlying device type (for NIRC2 this is DGH or Lakeshore). Thus, detection of errors, and responses to those errors, will be done on a device-specific basis. For read failures of NIRC2 digital signals, an error will result in a return value of -1.

Note that the configuration file (discussed in §3.4.2) will provide for i/o direction and active states. The i/o direction will specify which signals are inputs and which are outputs, on those devices which provide for configurable ports. The active states allow negative logic to be hidden, so that users can assume 1 means on and 0 means off. The digital input and output values will be or'd with the active state bits.

**3.4.1.2.1 Writing a digital output.** The function for outputting a value to a digital signal or entire port will take as parameters a device number, a channel number, and an output value. A channel value of -1 indicates that the output is for an entire port, in which case the output value will be assumed to be a bit field for all the output signals on the device.

The outputs will be exclusive or'd with the active state bits prior to being issued to the physical device.

The response will simply be the completion status of the command where -1 will indicate an error.

**3.4.1.2.2 Reading a digital output.** The function for inputting a value from a digital output signal or entire port will take as parameters a device number, and a channel number. A channel value of -1 indicates that the input is for an entire port.

The inputs will be exclusive or'd with the active state bits prior to being issued to the physical device.

The return value will be -1 on an error.

Note that not all digital output devices provide readbacks in which case an error will be returned.

**3.4.1.2.3 Reading a digital input.** The function for inputting a value from a digital input signal or entire port will take as parameters a device number, and a channel number. A channel value of -1 indicates that the input is for an entire port.

The inputs will be exclusive or'd with the active state bits prior to being issued to the physical device.

The return value will be -1 on an error.

### 3.4.1.3 Device information function

The device existence function will be called internally by the keyword library to verify that the arguments to many of the keywords are valid.

Initially there will be no keywords tied to this function but a support program may be added.

#### **3.4.1.4 User variable manipulation functions**

A set and a get function will exist for manipulating user keywords (also referred to as user variables, discussed in §3.2.1).

The set function will take <name,value> string pairs.

The get function will take a name string and will return a value string. The caller will be required to free the memory for the value string (standard rpc requirement).

#### **3.4.1.5 Device controller i/o functions**

A function will exist for sending arbitrary device controller command strings, and another for sending arbitrary device controller request strings. The intent is to provide a mechanism by which an expert can use the device-specific language, the assumption is that the expert is cognizant of device-specific commands and device-addressing requirements.

In all likelihood the two functions will be “telldevice” and “askdevice”. Askdevice will wait for a reply from the device controller, while telldevice will not. Telldevice will not cause a failure if a user issues a string to which a reply occurs, the reply will simply be ignored. Askdevice will eventually timeout on a lack of reply which may take several minutes, so the expert should be knowledgeable of the device commands.

Both functions will require as arguments a device number in addition to the command/request string.

These functions will provide for support programs discussed in §3.3.3.

### **3.4.2 Configuration files**

When the I/O daemon initializes it will read and parse the records contained in a configuration file. The config file will be identified to the I/O daemon as an argument when the daemon task is created. This config file is referred to as the i/o config file.

The i/o config file has three functions: to identify the communications links, to identify the devices connected to a given link, and to specify the characteristics of each of those devices.

More than one communications link may be specified but all devices and their characteristics, for a given link, must be specified before another link is specified. That is, all device information following a communications link is assumed to be relevant to that communications link, up to the point that another comms link is specified.

#### **3.4.2.1 Communications link specification**

The purpose of the comms link spec is to identify the comms device on which one or more device controllers are connected.

The device controllers may be connected to a host (Unix machine) serial port, or to a terminal server port. The syntax of the communications link spec will indicate the type of the connection such that the I/O daemon will be able to derive the relevant information.

#### **3.4.2.2 Device specification**

For each device connected to a communications link there will be a device specification followed by several device characteristic specifications.

The device specification will contain the information provided in Table 16.

**TABLE 16. Information for device specification.**

<b>Device info</b>	<b>Description</b>
major device number	the instrument-wide number which must be an instrument-wide unique i/o device number.
minor device number	the device address or number to which a specific device controller, on the comms link in question, will respond. This is typically a device controller configuration issue. Note there may be multiple devices with the same minor device number, provided they are on separate communications links, thus the need for the aforementioned major device number.
intercharacter delay	used when writing to the comms link, if a delay is not specified then characters are output in bursts such that the controllers may not keep up (recommended value is 1 msec between chars).

### **3.4.2.3 Device characteristics**

Each i/o device will possess a set of characteristics which will be specified in the config file. The exact set required depends upon the device controller and its programming (NIRC2 i/o device controllers are currently DGH and Lakeshore modules).

Each device characteristic will be on a separate line in the config file.

If a characteristic requires a string of multiple words then that string will be enclosed in double quotes.

How the individual characteristics are handled is a function of the specific device controller requirements (i.e., a DGH controller has a different command set from a Lakeshore controller). As mentioned in §3.3, there will be a specific software module for each type of controller; those modules will be responsible for parsing and retaining the pertinent motor characteristics.

All characteristics are defined in Table 17 even though some of them are currently not needed (hooks have been added for other controllers such as Lakeshore and Xycom).

**TABLE 17. Device characteristics.**

Characteristic	Description
device type	<p>this will be one of DIN, DOUT, DIO, ADC, or DAC and will be required to be the first item on the first device characteristic line for a given device. For DIN, DOUT, and DIO, a hex value must follow.</p> <p>For DIO the hex value will specify the channels' I/O directions. For DIN and DOUT the hex value will specify the channels' active states.</p> <p>All other characteristics are pertinent to analog i/o devices.</p>
channel number	<p>this will be a required value for the analog input or output. All characteristics will be associated with the specified channel up to the point that a new device type is encountered.</p>
scale	<p>this will be an optional scaling value for an analog input or output. The scale will default to 1.0.</p> <p>this will be an optional offset value for an analog input or output. The offset will default to 0.0.</p>
breaktable	<p>this will be optional and will indicate that a list of value pairs, on separate lines, will follow. The list will constitute a lookup table that will be used for the analog channel specified by channel number. The end of the table will be delineated with 'end' on a line by itself.</p>

### 3.4.3 Support programs

Support programs will exist for the purposes of troubleshooting, integration, and I/O module setup. These programs will be intended to be used by the instrument software specialist and/or the instrument specialist, but not the user community.

The support programs bypass the keyword library and interface directly to the I/O daemon via the rpc interface. Thus, they will all be stand-alone rpc clients which utilize the rpc functions discussed in §3.4.1.

The support programs are software modules written in the C programming language.

The I/O daemon support programs are enumerated in Table 18.

**TABLE 18. I/O daemon support programs.**

<b>Program</b>	<b>Description</b>
askdevice	will allow one to issue a command string to a specified device module/controller for which a response will be expected. Multi-word strings will be expected to be enclosed in double quotes. The user will be expected to be knowledgeable of the device language (DGH or Lakeshore).
telldevice	will allow one to issue a command string to a specified device module/controller, for which no response is expected. Multi-word strings will be expected to be enclosed in double quotes. The user will be expected to be knowledgeable of the device language (DGH or Lakeshore).
devprop	will allow one to set a specified device property variable to a specified string or to get the current value of a specified device property variable. The variables are essentially user keywords/variables (discussed in §3.2.1) that will be retained in a hash table within the I/O daemon.
devinfo	will allow one to get a display of the device module characteristics of one or all device modules/controllers.

#### **3.4.4 Logging**

I/O daemon logging will be performed by syslog calls embedded throughout the daemon functions. Syslogging is discussed in §4.2.

The logging level is controlled by the environment variable LOG\_UPTO. This env var must be set within the context in which the I/O daemon is launched, in order to affect syslogs from the daemon.

The default logging level will provide for syslogs of errors encountered in the I/O daemon. Setting the logging level to 'debug' will provide a logged message for all entry and all successful exits of rpc calls (client interface) and all I/O specific calls (DGH directory), and reading of config files.

An environment variable will be provided to allow enabling debug level logging of the device command strings and replies, from within the low level send and receive functions.

An environment variable will be provided to enable writing the device command strings and replies to the window in which the daemon was launched, from within the low level send and receive functions

#### **3.4.5 I/O Device controllers**

The NIRC2 contains several thermistors as well as analog and digital signals for temperature and coldhead control. The temperature control signals will be interfaced to a Lakeshore unit, whereas the remaining signals will be interfaced to several DGH digital and analog i/o modules. Each of these units provides a command set for reading and setting values over a serial link.

As the details of the Lakeshore controller are not known at this time, the intent is to connect it to a separate serial link. This will not be a problem as the I/O daemon design permits for communicat-

ing over multiple links. Note that hooks have been added to the I/O daemon to accommodate communications with the Lakeshore controller.

The DGH modules are configured for the RS-485 serial protocol which allows for connecting multiple controllers to any given serial link. As the modules are physically located in two separate locations (temperature control at the instrument and coldhead control in the computer room), there is a need for at least two communications links. In fact, we intend to connect the motor controllers to terminal servers, thereby eliminating any long cables other than the existing ethernet cables. At the instrument this will be the same terminal server used by the motor controllers (§3.6.1).

#### **3.4.5.1 Device addressing**

The multi-dropping of device controllers on a given serial link requires that each controller on a given link must have a unique address. This is provided for within the DGH controllers' programming language and is the minor device number discussed in §3.4.2.2 Table 16).

#### **3.4.5.2 Device setup**

When the DGH controllers are received from the manufacturer they are programmed with a default setup, in terms of device address and serial communications (e.g., baud rate). The default setup precludes a controller from being multi-dropped with other controllers. Procedures will be established for the initial configuration of the DGH controllers.

### **3.4.6 Temperature Control**

Temperature control will be handled by the Lakeshore controller. Keywords will be provided for control and status (enumerated in Table 6). A Unix shell script which will use the keywords will be written for monitoring the temperature control.

### **3.4.7 Coldhead Control**

Coldhead control will be via the DGH i/o modules. The control will provide for a fast and a slow speed. Keywords will be provided for control and status (enumerated in Table 6). A Unix shell script which will use the keywords will be written to monitor and control the coldhead. The resulting process will be performed at a default rate of once per hour, however, the instrument specialists will be able to adjust the rate.

## **3.5 Pupil Tracking Control**

A pinion turns all four of the rotating pupil masks as shown in Figure 4. Given the keywords

OBRTSKY - the physical angle of the AO K-mirror in units of degrees on the sky (i.e., twice the shaft rotation)

EL - telescope elevation

the equation for the demanded physical position of the pinion is

$$c_1 + c_2(\text{obrtsky} + \text{el})$$

Physical Rotator Position

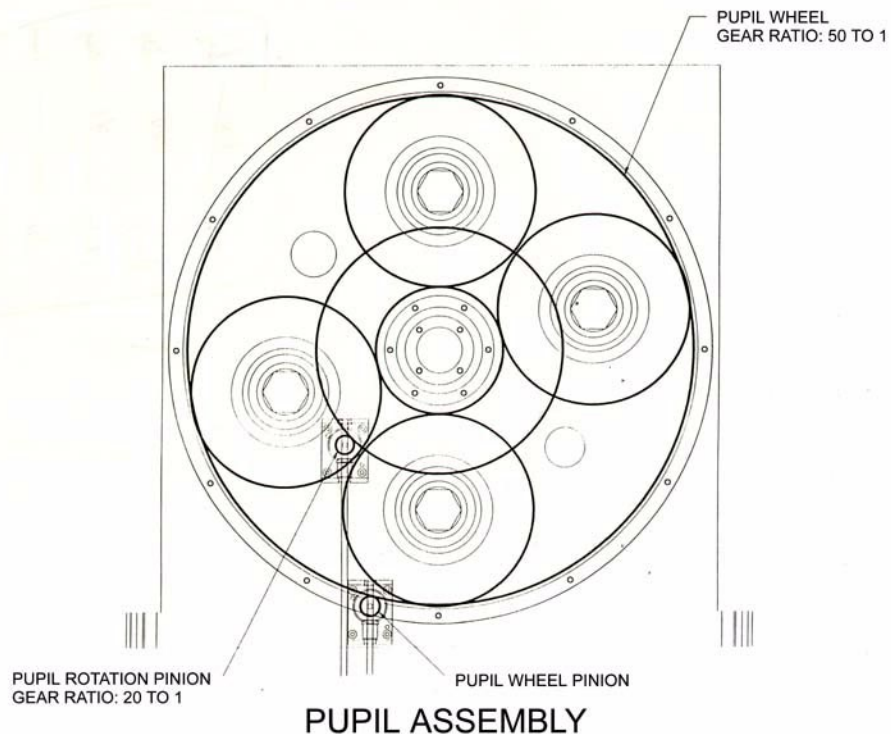
(EQ 1)

where  $c_2$  is the 20:1 gear ratio and  $c_1$  is the offset needed to align the hexagonal mask with the hexagonal primary mirror.

The Unix process *prot* monitors the two keywords, OBRTSKY and EL, and moves the pinion as per equation1 by writing the NIRC2 keyword PRDWVEL (Table 5). The *prot* event loop will be modeled after the watch\_imrot task and will hence inherit the error checks, reconnect logic, and logging that was developed and tested for that task.

The value written to PRDWVEL will be the velocity computed using the time stamps provided by the two positional values EL and OBRTSKY. Given maximum acceleration of

$1.5 \times 10^{-5}$  radians/sec<sup>2</sup> and a one milliradian accuracy, *prot* must update velocity at 0.1 Hz. On the existing HIRES system at Keck, we update rotator velocity at 0.5 Hz, so 0.1 Hz should be achievable using the same event loop.



**FIGURE 4. Pupil Assembly**

### 3.6 UCLA/NIRSPEC Software for Reading the Detector

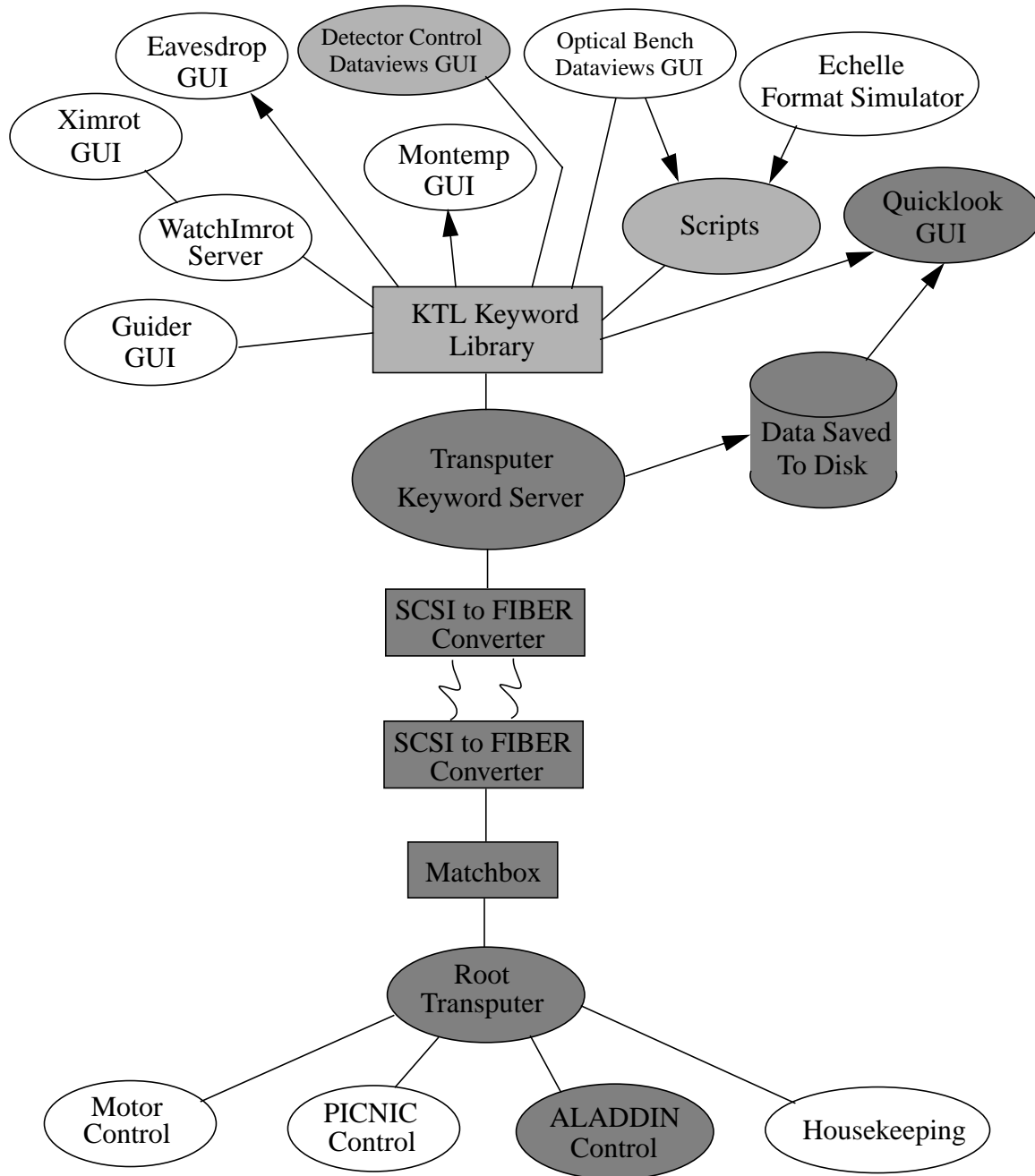


FIGURE 5. NIRSPEC Software. Shading reflects degree of NIRC2 inheritance.

All software for control of the ALADDIN detector array will be inherited from the UCLA/NIRSPEC system. The NIRSPEC software has previously been reviewed at all levels (i.e., PDR, CDR...) and will be given a readiness review in preparation for final shipping on January 15, 1999. Only a small fraction of the NIRSPEC software will be used for the NIRC2 instrument. Figure 5 shows the block diagram for the complete NIRSPEC system, with the inherited portion

shaded gray. The degree of shading reflects the degree of inheritance. Figure 1 in §3.0 gives the block diagram of the merged NIRSPEC/NIRC2 software system as it is currently envisioned. One of the most important aspects of the inherited system is its proven ability to work well with the UCLA electronics also being inherited for the NIRC2 instrument. In the following subsections, we detail the components of the inherited UCLA software and the modifications required for NIRC2.

### 3.6.1 Keywords

The core of the NIRSPEC software is an RPC server and KTL keyword library consisting of more than 200 keywords. This system is complete in the sense that it is fully capable of controlling any aspect of the NIRSPEC instrument, and can interact with all GUI's and other user interfaces. It is also fully compatible with UNIX scripting using the standard KTL interface described in §3.1. In fact, several similar scripts already exists for the NIRSPEC instrument (e.g., snap and bxy9). This server and KTL library are fully detailed within the NIRSPEC Programming Notes NSPN 06.01 (Keyword Library), NSPN 07.01(RPC Server), NSPN 27.00 (Server Code File Locations and Descriptions), and NSPN 29.00 (Adding Keywords).

The primary modification to the keyword library for NIRC2 is to dramatically reduce the number and scope of the keyword library to only deal with the ALADDIN detector array, its data and the quicklook client (§3.6.2). None of the mechanism control, temperature monitoring, PICNIC detector control, DCS interaction, or other housekeeping keywords will be maintained for NIRC2. Below is the complete list of inherited keywords:

---

go	Begin an exposure and save the result in a data sequence.
test	Begin an exposure, but only create a temporary FITS file.
savetest	Save current test frame.
abort	Abort an exposure.
framerdy	Specifies when data is fully retrieved.
frameend	Indicates when data is written to permanent file.
coadds	Define the number of coadds.
itime	Define the exposure time.
rootname	Define the root portion of the FITS file name.
filenum	Set the number to use in the data sequence for the next file
sampmode	Specify either single, CDS, or Fowler sampling.
multispec	Define the number of Fowler samples to use.
dispname	Filename for a disk file to be displayed with Quicklook.
display	Signal the Quicklook to display the disk file specified.
telescop	Set the telescope name.
observer	Set the observers' name.
outdir	set the output directory name.
dcshaders	Get DCS headers for FITS files.

---

---

fileovwr	Set file over-write flag.
filename	Alternate filename if not overwriting.
object	Set the object name.
comment	Put a comment into the header.
detbias	Set the DAC output for the detector bias.
q1offsetspec	Specify the preamp offset voltage for quadrant 1.
q2offsetspec	Specify the preamp offset voltage for quadrant 2.
q3offsetspec	Specify the preamp offset voltage for quadrant 3.
q4offsetspec	Specify the preamp offset voltage for quadrant 4.
freq.spec	Set the filter frequency in the preamps.
gain.spec	Set the preamp 2nd stage gain.
tsptrace	Set Transputer link trace flag (diagnostic).
tspcid	Set a command id to send over TSP link.
tspparam	Send a transputer parameter to tspcid.
tspack	Acknowledgment signal from transputers.
tspmsg	Message received from the transputers.
tspstat	Status message sent back from transputers.
tspupdate	Request to read a transputer value.
tspreset	Reset the TSP link.
cid.test	A test link.

In addition to these keywords, several new ones will be added to control the NIRC2 specific aspects of the detector and electronics:

---

xsize	Size of the x-dimension of the subarray
ysize	Size of the y-dimension of the subarray.
poffset	Offset voltage for the entire array for the additional preamp.

### 3.6.2 Quick Look

The UCLA/NIRSPEC image viewer will also be inherited. It is programmed in IDL and uses the KIDL package (developed by CARA) to connect to the KTL server. It monitors the server and automatically displays images as they become available. It has a complete set of basic data reduction and analysis features including: image arithmetic, circular photometry, image statistics, hori-

zontal and vertical cuts, surface plots, and gaussian fitting. The quicklook is detailed in the NIRSPEC programming document NSPN 13.00. There are no significant modifications required for the NIRC2 system.

### 3.6.3 Occam Transputer Software

OCCAM is the programming language for the UCLA transputer-based electronics system. The language supports true multiprocessor, parallel processing, and is a high-level language with a rich command structure. Just as described in the server above, only a small fraction of the NIRSPEC Occam code will be used in the NIRC2 system. The complete set of Occam code, plus a general introduction to Occam programming is given in NIRSPEC programming notes: NSPN 08.00 (Host-Transputer Communications), NSPN 17.00 (Root Transputer & Housekeeping), NSPN 18.00 (clocking), NSPN 19.00 (Data Acquisition), NSPN 20.00 (Motion Control Transputer Code), NSPN 21.00 (Transputer-Transputer Communications), NSPN 22.00 (Introduction to Transputers and OCCAM Programming), NSPN 26.00 (Transputer Program Building and Code Management), NSPN 31.00 (Transputer Keywords).

### 3.6.4 Subarray Clocking

Perhaps the most complicated modification to the UCLA software is the addition of subarray clocking for the ALADDIN array. This is being added to allow rapid frame sequences, and will be carried out by UCLA personnel. The changes involve both OCCAM modifications and changes to the server. The required changes include: Fast clocking through unused pixels, rearranging the data as it is currently stored, packaging the smaller data set differently for transfer to the server, correctly unpacking arbitrary subarrays, saving the data in the proper FITS format, and creating keywords to define the array size. These changes are non-trivial but straightforward given the variety of clocking patterns and array sizes already supported.

### 3.6.5 ALADDIN 3 Modifications

An additional concern with the software is the likelihood that NIRC2 will actually operate with the new ALADDIN3 multiplexors instead of the current ALADDIN2 multiplexor used in the NIRSPEC instrument. This new multiplexor is being used with the last 6 devices from the PAIDAI consortium's 14 arrays. They should have several advantages including less row-to-row variations and more uniformity. They also have an identical pinout pattern, so the electronics doesn't need to be changed. Unfortunately, they do require a slightly different clocking scheme. The full description is given in the Raytheon CDR document DM 206CDR. The most significant effect is the requirement to overclock two of the quadrants to read out the last two rows of the other two quadrants. Once the data is stored in the electronics, there should be no other changes necessary. These changes will also be completed by UCLA personnel.

### 3.6.6 FITS Writer Modifications

As described in D.3, every Keck keyword library provides a subroutine to read an individual keyword (`ktl_read`). In addition, a keyword library can provide an alternate method for reading all of the keywords suitable for a FITS header in a single call (`ktl_ioctl(KTL_HEADER)`).

Historically, Keck FITS writers have used one of two methods to collect FITS keywords:

1. *ala carte* - Read each keyword individually via `ktl_read`.

2. *oneshot* - Read all keywords in one shot via `ctl_ioctl(KTL_HEADER)`.

There are pros and cons to each of these methods as follows (some existing Keck FITS writers offer a choice).

- advantages of *ala carte* over *oneshot*:
  1. can be driven by a client-side configuration file.
  2. less work required to produce the keyword library.
- advantages of *oneshot* over *ala carte*:
  1. usually more efficient.
  2. timeouts better handled.
  3. less work required to produce the FITS writer.

For NIRC2, given the distribution of development across several sites, the advantage of number 3 above is amplified. Thus the `ctl_ioctl(KTL_HEADER)` feature will be provided within the NIRC2 mechanism keyword library. The content of the block of keywords "suitable for a FITS header" will be driven by a configuration file called `nirc2_header_info` (following the model of the DCS, HIRES, LRIS, and NIRC1 keyword libraries).

The only changes needed for the software inherited from the UCLA NIRSPEC system are:

- making a call to `ctl_ioctl(KTL_HEADER)` and
- writing the contents of the returned arrays in the same way that the detector keywords are currently being written to NIRSPEC headers.

## 4.0 Software Environment

### 4.1 Desktop

Observers will use one of 20 revolving computer accounts (k2nirc1 through k2nirc20). Each account will be refreshed periodically with a master account template to provide a stable, consistent starting point for each run. The master template will run the tcsh and have the normal Unix structure, complete with .login, .cshrc, .openwin-menu, .netscape, etc. files and directories.

The PATH and IDL\_PATH environment variables will be set to allow observers access to all necessary executables, observing scripts, IDL, and IDL procedures. IRAF will also be available via the command line. Scripts will be organized into a small number of well-defined groups, easily allowing access to appropriate commands or restricting access to other commands. The observing accounts will specifically exclude some engineering level functions and scripts from their PATH.

As with NIRC1, the PATH environment variable will be used to distinguish between a single, primary command window and multiple auxiliary command windows. While the primary command window provides access to all observing commands, auxiliary command windows are restricted to commands that will not disrupt data acquisition. So, for example, commands that move filter wheels are not available in auxiliary command windows. I/O in the primary control window will be logged to a file.

Other windows will include xshow status display(s) (e.g. Figure 3), the event log ("log tail" in Figure 2), and the UCLA quicklook GUI.

An engineering account, k2nirceng, will include all of the above, plus easy access to various engineering functions, scripts, and displays. It will also have more direct access to various engineering logs, etc. Developmental programs, scripts, and environments may also be maintained from this account. As a result, it will NOT be refreshed periodically as the observing accounts will.

The philosophy behind the observing accounts is to provide easy access to all of the normal functions required during observing. With this goal in mind, the OpenWindow menu will provide single-click access to the following functions:

- Start (start up and configure data-taking software, initialize hardware as needed, and initialize observing windows). A prompt will allow configuration to either a nominal state or a previously saved state.
- Close windows (close all displays and observing windows without disturbing the data-taking software; this would be used, for example, when moving from one computer to another)
- End night (configure instrument for end of observing, shut down hardware and data-taking software as necessary, close observing windows; this would be used at the end of a night's observing)
- Start/Restart various individual components (including status display, observing windows, windows showing log files, the quick look display)
- Help (Netscape; the home page will be the NIRC2 home page at CARA, with relevant documentation and links to other Web sites)

- Telescope/AO displays (guider eavesdrop, telescope status displays, both text and graphical, AO displays)
- Weather (the XMET weather display)

## 4.2 Logging and alarms

As alluded to in previous sections (3.2.7, 3.3.4, 4), the logging of informational messages, keyword changes, and errors will all be done via the Unix log system.

When a function performs a syslog call a 'message' is passed to the syslog daemon, namely syslogd, (there are other possibilities, defined in the Unix man pages, that one may wish to utilize in some debugging situation, but here we are only concerned with the syslogd). The syslogd formats an error message which it then writes to the syslog file.

The syslog file is defined in /etc/syslog.conf. The default syslog file, for NIRC2, will be /var/log/local0, however, it will be possible for this to be changed.

The logging consists of several levels/priorities. The level is controlled by the environment variable LOG\_UPTO, which is normally not defined, such that all priorities above LOG\_DEBUG are logged. It will be necessary to set LOG\_UPTO within the context in which the NIRC2 command (keyword modification) will be executed. For instance, if one were to use 'show' and wants the log level increased to debug level, then one would need to 'setenv LOG\_UPTO debug' in the window in which the show is to be performed.

The use of syslog provides a log of the errors generated by the NIRC2 software, but does not alert the user to those errors. Error notification will be provided by a tool, used on many of the current Keck instruments, namely, tklogger. Tklogger monitors the syslog file and "pops-up" an error window when an error is logged, for which tklogger has been configured to detect. Tklogger is driven by a configuration file, providing an instrument specialist with the flexibility to specify which errors will alert the users, indicate the possible causes of errors, and suggest courses of action for the user to take.

### 4.2.1 Thermal Logging

Keywords will be provided for all the thermistors, and temperature and coldhead controller statuses (enumerated in Table 6). A Unix script will be written to read and store the values of the keywords at a default rate of once every 15 minutes. The instrument specialists will be able to adjust the rate.

The thermal keywords will conform to the keyword-changed notification technique discussed in §3.2.7. Thus, the values will automatically update within any xshow windows in which they appear.

## 5.0 Software Test Plan

As seen in section 6.0, Schedule, we plan two software releases: a release that provides the functionality needed for laboratory integration, and a second complete release with the functionality needed for observing with NIRC2 at Keck.

## 5.1 Testing the LAB I&T Release

Prior to the laboratory integration and test (I&T) release, we will use show, modify, and xshow (Appendix D.2) to unit test

- motor keywords by driving the animatics motor available in Waimea
- DGH keywords by driving the DGH module available in Waimea
- Lakeshore keywords by connecting a serial line to a laptop in Waimea
- Alladin keywords by taking images at UCLA

### 5.1.1 Testing Motor Keywords

We will simulate limit switches, brakes, etc., with toggle switches and LED readouts. We will test shaft movement by simply attaching a vane and observing its position. Although these methods will help us prepare for the site visit (see row 3 of Figure 6), accelerations profiles, etc., will require testing on site with the real mechanisms.

### 5.1.2 Testing DGH Keywords

We will use LED readouts or an oscilloscope to confirm TTL levels are output on the lines that will eventually be connected to the cold head speed controls, AO lamps, etc.

### 5.1.3 Testing Lakeshore Keywords

We will test Lakeshore keywords by connecting a serial line to a laptop and simulating the data streams expected and produced by the real Lakeshore unit. These keywords will be the most difficult to test in Waimea. Not only do we not have a unit to test with, but, as of this writing, the manual defining the Lakeshore command protocol is unavailable.

### 5.1.4 Testing ALADDIN 3 Keywords

We will test Alladin keywords by repeating the tests used for NIRSPEC.

### 5.1.5 Scripts

Although a few of the scripts needed for lab I&T (§3.1.1 and §4.2.1) will be tested prior to the site visit (see row 3 of Figure 6), most of the scripts for lab I&T will be developed and tested during the visit.

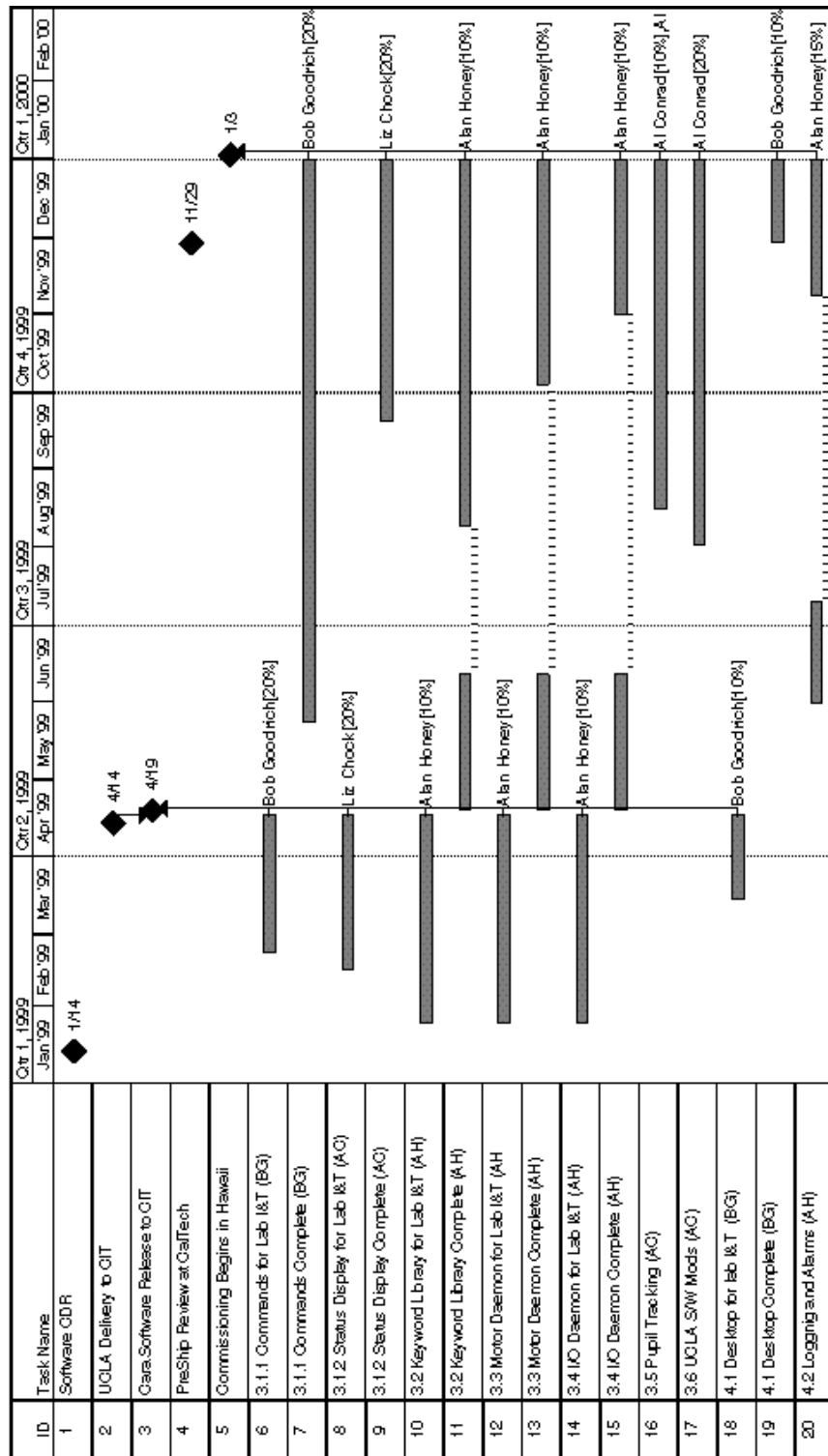
## 5.2 Testing the Complete Release

Software testing for the complete release will involve full up testing in an environment simulating actual observing. This will require use of the DCS telescope simulator. This test will consist of running all of the scripts enumerated in Table 1.

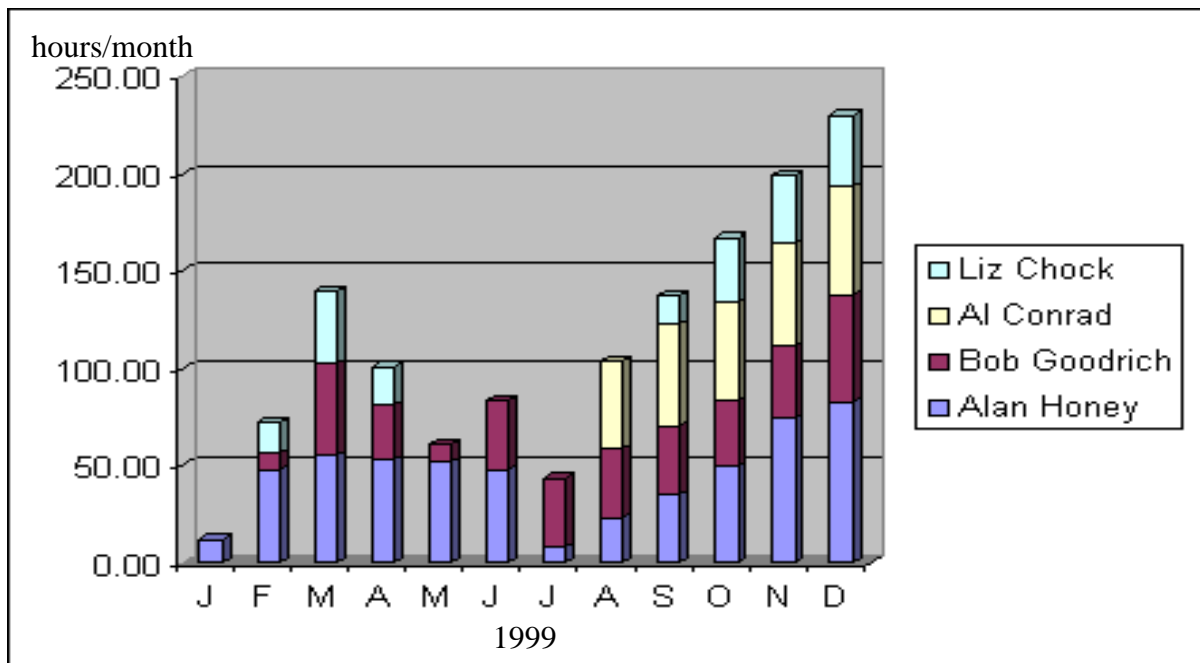
## 6.0 Schedule



FIGURE 6. Schedule



**FIGURE 7. Workload**



## Appendix A. Design Goals

In general, the design goals of the NIRC2 software team were to meet all the requirements specified in the NIRC2 Software Functions Requirements (NIRC2-98-001), as well as address the issues discussed in the review board's report from the NIRC2 Software CDR of October 1997.

The cross reference between the software functions requirements and this design document are provided by Appendix B Table 19. Review of the table indicates that the requirements have been satisfactorily met, implying that the design did in fact "flow from the requirements."

### A.1 Simplicity

The design attempts to be at least as simple as other Keck instruments. This statement is supported by the fact that the design is similar to, and to a large degree derived from, the NIRC1 P3 and LWS software.

The simplicity was reinforced by the use of commands and tools provided by the Unix operating system, as opposed to creating an instrument-specific command line interpreter.

### A.2 Modularity

Modularity has been addressed by structuring the software to be consistent with the architecture. Five major divisions exist within the CARA-provided software: the shareable keyword library, the motor daemon, the I/O daemon, pupil rotator control, and a set of Unix shell scripts which use the keyword library to provide higher level functions to the users.

Modularity within the keyword library was attained by recognizing that three categories of keywords exist (observing, maintenance, and engineering as discussed in §3.2) such that the software can be structured into level motor functions to support the maintenance and engineering keywords, and higher level 'mechanism' functions to support observing keywords. In addition, the higher level mechanism functions can be divided into three distinct types (wheel type mechanisms, such as filters; translational type mechanisms, such as camera slides; rotator type mechanisms which consists of only the pupil rotator) such that a distinct software module can be written for each type.

Modularity within the motor and I/O daemons was attained by recognizing that each daemon must: provide an interface consisting of a set of RPC functions, provide a means of connecting to serial and/or ethernet links and provide communications on those links, provide for linking in device-specific modules containing functions for handling the module-specific language syntax and behavior.

Note that the five major divisions of the architecture allowed each division to be worked on independently, although complete testing could not be done independently in all cases.

### A.3 Reliability

Reliability was addressed by: reusing software which has been extensively used on other instruments; adhering to Keck software standards and programming practices (KSDs 3, 11, 50, 66, 69, 88, 116); use of a code management system (CVS); use of Keck build/make infrastructure; and use of a software run-time tool (purify) which identifies run-time errors and memory leaks.

## **A.4 Testability**

Testability of the software was augmented by both the architecture and the modularity. The testing was not totally independent as the observing scripts and pupil rotator control required a functional keyword library which, in turn, required the motor and I/O daemons to be functional.

Once the motor daemon was functional it was possible to test the low-level motor control via the support programs and then by the maintenance and engineering keyword.

Once the motor daemon was functional it was possible to test keywords associated with the analog and digital i/o signals by interconnecting the inputs and outputs between device modules.

Not all the observing scripts and modes can be properly tested until the instrument is physically assembled. However, the built-in simulation, keyword change notification, and logging facilities allowed confirming that the motor and device commands as well as behavior are as expected.

## **A.5 Extendibility**

Extendibility is addressed by: designing into the daemons the ease of adding new controllers, by writing modules for those controllers and linking them into the respective daemon; extensive use of configuration files to define the keywords and hardware; providing for user-defined keywords; and use of a Unix scripting language and editors.

## **A.6 Error Checking**

### **A.6.1 Function Returns**

All function status returns will be checked and passed up the calling hierarchy. The status will be reported to the user via the ktl\_set\_errtxt and syslog methods used by other instruments at Keck.

### **A.6.2 File I/O Errors**

File writing errors caused by permission or space problems will be trapped and handled using the techniques developed on other instruments at Keck (i.e., trying alternate directories).

### **A.6.3 Time Outs**

All motor moves will provide time out errors tailored to the expected latency of the device.

## Appendix B. Requirements Cross Reference

TABLE 19. Requirements cross reference.

Requirement Number	Design Book Section	Comments
2.1.1	A.2, §2.4	
2.1.2	Figure 1	
2.1.3	A.5	
2.1.4	§2.2	RPC, IDL, standard unix tools
2.1.5	§1.1.2	NIRC1 performance sufficient
2.2.1	A.4	
2.2.2	§3.2.9., 3.3.5	
2.2.3	-->	DCS simulator
2.3.1	A.3	
2.3.2	§3.2.7, 3.3.4, A.6	
2.3.3	§3.2.5.1	
2.3.4	A.3	
2.3.5	A.4, §5.0	
2.3.6	§3.3.4	
2.4.1	A.3	adhering to KSD 3
2.4.2	A.3	adhering to KSD 3
2.4.3	A.3	adhering to KSD 3
2.4.4	A.3	adhering to KSD 3
2.4.5	A.3	adhering to KSD 3
2.4.6	A.3	adhering to KSD 3
2.4.7	A.3	adhering to KSD 3
2.4.8	A.3	adhering to KSD 3
3.1.1	-->	Keck infrastructure
3.1.2	Figure 3	
3.1.3	Table 1	
3.1.4	Table 1	
3.1.5	§3.6.2, Table 1	buf1
3.1.6	Table 1	e.g., mxy
3.1.7	§3.1.2	use of shell scripts
3.1.8	§3.1.2	use of Unix editors
3.1.9	Table 1	help
3.2.1	§4.1	paragraph 4
3.2.2	§3.2.6, 3.3.2, 3.4.2	configuration files

**TABLE 19. Requirements cross reference.**

Requirement Number	Design Book Section	Comments
3.2.3	§3.2.2, 3.2.3, 4.1	4.1 paragraph 4
3.2.4	§3.2.6, 3.3.2, 3.4.2	configuration files
3.3.1	§3.6.1	e.g., sampmode, xsize, ysize, etc
3.3.2	§3.6.1, Table 1	readmode, subc
3.3.3	§3.6.6	
3.4.1	§3.6.6	
3.4.2	§3.6.2	
3.4.3	-->	NFS as per existing keck instruments.
3.4.4	§4.2.1	temperatures only
3.4.5	§3.1.1, 3.2.7	
3.4.6	§3.2.7, 3.3.4, 3.4.4, 4.2	
3.5.1	§4.1	
3.5.2	Table 1, §4.1	Table 1: savestate, loadstate. 4.1: start
3.5.3	§4.0	
3.5.4	Table 1, §4.1	Table 1: savestate, loadstate. 4.1: start
3.5.5	A.3	
3.5.6		A.6
4.1		unix rlogin mechanism
4.2		unix password mechanism
4.3	§4.1	k2nirceng versus k2nirc1-10
4.4	§4.1	paragraph 3
4.5	§4.1	paragraph 3
5.1.1	§3.1.1	
5.1.2	§3.1.1	unix ';' ;'
5.1.3	§3.1.1	csch
5.1.4	§3.1.1	tcsh
5.1.5	-->	not in current design ("desirable")
5.1.6	§3.1.3	
5.2.1	§3.1.3, Figure 3	
5.2.2.1	§3.1.3, Figure 3	e.g., "-fn 10x20"
5.2.2.2	§3.1.3, 4.2	tklogger
5.2.2.3	§3.1.1, 3.1.3	-fn 10x20 tested
5.2.2.4	§3.1.1, 3.1.3	-fn 10x20
5.2.2.5	§4.2	tklogger

**TABLE 19. Requirements cross reference.**

<b>Requirement Number</b>	<b>Design Book Section</b>	<b>Comments</b>
5.2.2.6	§3.1.3	xshow is user configurable
5.2.2.7	§3.1.3, Figure 3	
5.2.2.8		
5.2.2.9	Table 10 (units)	
5.2.2.10	Table 10 (datatype)	
5.2.2.11	Table 10 (precision)	
5.2.2.12	Table 10 (datatype)	
5.2.3.1	Figure 3	
5.2.3.2	D.1 and D.2	all keywords can be xshowed
5.2.3.3	§3.1.1	
5.2.3.4	-->	existing tool xcount
5.2.3.5		
5.2.4.1	§3.2.8	
5.2.4.2	Table 4, "NAME"	
5.2.4.3	N/A	
5.3.1.1	§3.6.2	
5.3.1.2	§3.6.2	
5.3.1.3	§3.6.2	
5.3.1.4	§3.6.2	
5.3.1.5	§3.6.2	
5.3.2.1	§3.1, Table 1	UNIX "cp", wd
5.3.2.2	§3.1, Table 1	buf1, odiff, pdiff, sdiff
5.3.2.3	N/A	
5.3.2.4	§3.1, Table 1	bstat, ostat, pstat
5.3.2.5	§3.1, Table 1	bstat, ostat, pstat
5.3.2.6	N/A	
5.3.2.7	N/A	
5.3.2.8	N/A	
5.3.2.9	N/A	
6.1.1.1.1	§3.1, Table 1	goi
6.1.1.1.2	§3.1, Table 1	go
6.1.1.1.3	§3.1, Table 1	Cntl-c
6.1.1.1.4	§3.1, Table 1	Cntl-c
6.1.1.2.1	§3.6.1	keywords sampmode and multispec

**TABLE 19. Requirements cross reference.**

<b>Requirement Number</b>	<b>Design Book Section</b>	<b>Comments</b>
6.1.1.2.2	§3.1, Table 1	frame, nextfile
6.1.1.2.3	§3.1, Table 1	readmode
6.1.1.2.4	§3.1, Table 1	detbias, coadd, subc, tint
6.1.2.1	§3.1, Table 1	ln2, lhetemp
6.1.2.2	§3.2.1, Table 6	
6.1.2.3	§3.2.1, 3.2.4	
6.1.3.1	Table 1	filter (generic), j,h,k (delivered filter complement to be determined)
6.1.3.2	§3.1, Table 1	blk* commands (delivered blocker complement complement to be determined)
6.1.3.3	§3.1, Table 1	fgr
6.1.3.4	§3.1, Table 1	fgr
6.1.3.5	-->	need specifics of commands mentioned
6.1.3.6	§3.2.1, Table 4	home
6.1.3.7	§3.2.1, Table 4	
6.1.4.1	§3.1, Table 1	camera
6.1.4.2	§3.2.1, Table 4	
6.1.4.3	§3.2.1, Table 4	
6.1.5.1	§3.1, Table 1	sgr
6.1.5.2	§3.1, Table 1	spupil
6.1.5.3	§3.1, Table 1	clrgrism
6.1.5.4	§3.1, Table 1	sgr
6.1.5.5	-->	will not ship with an echellette
6.1.5.6	§3.2.1, Table 4	home
6.1.5.7	§3.2.1, Table 4	
6.1.6.1	§3.1, Table 1	slt1
6.1.6.2	§3.1, Table 1	dekker
6.1.6.3	§3.1, Table 1	mskclr, sltclr, sltmclr, sltsclr
6.1.6.4	§3.1, Table 1	close
6.1.6.5	§3.2.1, Table 4	home
6.1.6.6	§3.2.1, Table 4	
6.1.7.1	§3.1, Table 1	pupil
6.1.7.2	§3.1, Table 1	pupil
6.1.7.3	§3.1, Table 1	pupil
6.1.7.4	§3.1, Table 1	pupil

**TABLE 19. Requirements cross reference.**

Requirement Number	Design Book Section	Comments
6.1.7.5	§3.2.1, Table 4	home
6.1.7.6	§3.2.1, Table 4	
6.1.7.7	§3.5, Tables 4 & 5	
6.1.8.1	§3.1, Table 1	preslit
6.1.8.2	§3.1, Table 1	psltclr
6.1.8.4	§3.2.1, Table 4	home
6.1.8.4	§3.2.1, Table 4	
6.1.9.1	§3.1, Table 1	corona
6.1.9.2	§3.1, Table 1	sltclr
6.1.10.1	§3.1, Table 1	lamp
6.1.10.2	§3.1, Table 1	lamp
6.1.10.3	§3.1, Table 1	lamp
6.1.10.4	§3.1, Table 1	lamps
6.1.10.5	§3.1, Table 1	lamps
6.1.11.1	§3.1, Table 1	n, s, w, e, en
6.1.11.2	§3.1, Table 1	el, az, azel
6.1.11.3	§3.1, Table 1	mxy, x, xy, y
6.1.11.4	§3.1, Table 1	px, pxy, py
6.1.11.5	§3.1, Table 1	mov
6.1.11.6	§3.1, Table 1	foc
6.1.11.7	§3.1, Table 1	dfoc
6.1.11.8	-->	sky
6.1.11.9	-->	sky
6.1.11.10	§3.6.6	
6.1.11.11	§3.6.6	
6.1.11.12	§3.1, Table 1	ao2flat, ao2laser, ao2nat, ao2real, ao2sim
6.1.11.13	-->	rotmode = posang (sky)
6.1.11.14	-->	rotmode = vertical (sky)
6.1.11.15	-->	ao lamp control commands
6.1.11.16	§3.1, Table 1	closeon
6.1.11.17	§3.1, Table 1	dfoc; also normally done from SKY and/or AO commands
6.1.11.18	§3.1, Table 1	cent
6.1.11.19	§3.1, Table 1	scent

**TABLE 19. Requirements cross reference.**

<b>Requirement Number</b>	<b>Design Book Section</b>	<b>Comments</b>
6.1.11.20	§3.1, Table 1	sltmov
6.1.11.21	§3.1, Table 1	zero
6.1.11.22	§3.1, Table 1	mark and gomark
6.1.12.1	§3.1, Table 1	box5, box9, bxy5, bxy8, bxy9, mosaic, s5, sp2, sp55, sp56, sp74, thinxy
6.1.12.2	§3.1, Table 1	snapi, snapiv
6.1.12.3	§3.1, Table 1	foc3, foc5, foc8, malign
6.1.13.1	§3.1, Table 1	cdata, dir, runname
6.1.13.2	-->	Unix df command; disks
6.1.13.3	§3.1, Table 1	wd
6.1.13.4	-->	Unix cp command
6.1.13.5	-->	Unix mv command
6.1.13.6	-->	Unix rm command
6.1.14.1	-->	Unix sleep command
6.1.14.2	§3.1, Table 1	pause
6.1.14.3	-->	Unix at comand
6.1.15.1	§3.1, Table 1	observer
6.1.15.2	§3.1, Table 1	obj, object
6.1.15.3	-->	Unix editors
6.1.15.4	-->	Unix editors
6.2.1.1	§3.2.1, Table 4	
6.2.1.2	§3.2.1, Table 4	
6.2.1.3	§3.2.1, Table 4	
6.2.1.4	§3.2.1, Table 4	
6.2.1.5	§3.2.1, Table 4	
6.2.1.6	§3.2.1, Table 4	
6.2.1.7	§3.2.1, Table 5	
6.2.2.1	§3.1, Table 1	detbias
6.2.2.2	§3.1, Table 1	detoff
6.2.2.3	§3.1, Table 1	detoff
6.3.1.1	§3.2, Tables 7 & 8	
6.3.1.2	§3.2.3.2, Table 9 Table 15	ask askmotor
6.3.1.3	§3.2.3.2, Table 9 Table 15	tell tellmotor

**TABLE 19. Requirements cross reference.**

<b>Requirement Number</b>	<b>Design Book Section</b>	<b>Comments</b>
6.3.2.1	Table 4	raw
	Table 9	raw
6.3.2.2	Table 4, Table 9	delta
6.3.3.1	Table 6	
6.3.4.1	Table 6	
6.3.4.2	Table 6	
6.3.4.3	Table 6	
6.3.4.4	Table 6	
6.3.4.5	Table 18	telldevice
6.3.5.1	Table 1	lamp
6.3.5.2	Table 18	telldevice
6.3.6.1	Table 9	info
6.3.7.1	Table 9	tell
	Table 14	tellmotor
6.4.1	-->	show command
6.4.2	§3.1.2	csh allows variable # of parameters
6.4.3	§3.1.1	csh envvars
6.4.4	-->	all commands will echo appropriate information to users.
7.1.1	§3.3	
7.1.2	§3.3	
7.2.1.1	§3.3.3, Table 15	loadmotor
7.2.1.2	§3.3.3, Table 15	loadmotor
7.2.1.3	§3.3.6.1	
7.2.1.4	§3.3.6.3	
7.2.1.5	§3.3.6.3	
7.2.2.1	§3.3.6.3	
7.2.2.2	§3.3.6.3	
7.2.2.3	§3.3.6.3	
7.2.3.1	§3.3.2.3, Table 14	
7.2.3.2	Table 4 (EUP)	
7.2.3.3	Table 11	conversion factor
7.2.3.4	§3.2.1 Table 4	dest
7.2.3.5	§3.2.6.2	
7.2.3.6	Table 4	name = unknown, pos = -1

**TABLE 19. Requirements cross reference.**

<b>Requirement Number</b>	<b>Design Book Section</b>	<b>Comments</b>
7.2.3.7	Table 4	name = unknown, pos = -1
7.2.3.8	§3.2.5	
7.2.3.9	§3.2.5	
7.2.3.10	§3.5	
7.3.1.1	N/A	
7.3.1.2	N/A	
7.3.1.3	§3.4.7	
7.3.1.4	N/A	
7.3.2.1	Table 18	telldevice
7.3.2.2	Table 18	askdevice
7.3.2.3	§3.4.5.2	
7.3.2.4		** - AH comment: Does DGH retain state while powered up?
7.3.2.5	-->	keywords provide this functionality
7.3.3.1	Table 6	
7.3.4.1	Table 6	
7.3.4.2	Table 6	
7.3.5.1	Table 17 (scale)	
	Table 10 (units)	
7.3.5.2	Table 10 (units)	
7.3.5.3	Table 10 (units, precision)	
7.3.5.4	Table 10	config files don't change
7.3.5.5	§4.2.1	can write similar script for cold head keywords
7.3.5.6		** - Need to ask AH if possible.
7.4.1.1	N/A	
7.4.1.2	N/A	
7.4.1.3	§3.1, Table 1	lamp
7.4.1.4		** - needs clarification from KYM.
7.4.1.5	Table 6	
7.4.2.1	§3.1, Table 1	lamp
7.4.2.2	§3.1, Table 1	lamp
7.4.2.3	Table 6	
7.4.2.4	N/A	

**TABLE 19. Requirements cross reference.**

<b>Requirement Number</b>	<b>Design Book Section</b>	<b>Comments</b>
7.4.2.5	-->	this is an AO requirement
7.5.1.1	§3.4.6	
7.5.1.2	N/A	
7.5.1.3	-->	keywords don't change unless you set them; no specific lockout mechanism is provided.
7.5.2.1	Table 18	telldevice
7.5.2.2	Table 18	askdevice
7.5.2.3	-->	keywords don't change unless you set them; no specific lockout mechanism is provided.
7.5.2.4	-->	keywords, so this type of script possible
7.5.3.1	§3.1, Table 1	showtemp
7.5.3.2	Table 10 (precision)	
7.5.3.3	Table 10 (precision)	
7.5.4.1	Table 6	
7.5.4.2		Needs KYM clarification (BG: not software?)
7.5.4.3	-->	this is a Lakeshore requirement
7.5.5.1		Needs KYM clarification.
7.5.5.2		Needs KYM clarification.
7.5.5.3	-->	this is a Lakeshore requirement
7.5.5.4	-->	keywords, so this type of script possible
7.5.6	-->	keywords, so this type of script possible
7.5.7	-->	keywords, so this type of script possible
7.5.8.1	-->	keywords, so this type of script possible
7.5.8.2	-->	keywords, so this type of script possible

## Appendix C. Proposed NIRC2 Keywords

Please refer to §3.6.1 for the UCLA keywords.

**TABLE 20. Mechanism observing keywords.**

NAME	DESCRIPTION	TYPE	UNITS/VALUES	R/W FLAG
ARGONPWR	Argon_lamp_ctrl	boolean	true/false	write-only
BENCHSTPT	Opticalbench_T_setpoint	float	deg K	read/write
CAMDELTA	Camera_Slide_Offset_Cmd	integer	motor counts	write-only
CAMDEST	Camera_Slide_DestSts	integer	motor counts	read-only
CAMERA_T	temperature_diode_3	float	deg K	read-only
CAMEUP	Camera_Slide_Dest	float	eng units	read-only
CAMHOME	Camera_Slide_Home	boolean	true/false	read/write
CAMIDLE	Camera_Slide_IdleSts	boolean	true/false	read-only
CAMNAME	Camera_Slide_Name	string	posn name	read/write
CAMPOS	Camera_Slide_Position	integer	1 of n	read/write
CAMRAW	Camera_Slide_RawPosn	integer	motor counts	read/write
CAMSTAT	Camera_Slide_State		string	read-only
CAMTRGT	Camera_Slide_TargetPosn	string	posn name	read-only
CHEAD_ BENCH_T	temperature_diode_6	float	deg K	read-only
CHEAD_ DETECTOR_T	temperature_diode_7	float	deg K	read-only
CHEAD_ SHIELD_T	temperature_diode_4	float	deg K	read-only
DETBLCKT	temperature_diode_1	float	deg K	read-only
DETHEADT	temperature_diode_8	float	deg K	read-only
DETSTPT	Detector_T_setpoint	float	deg K	read-write
DETTEMP	detector_block_temp	double	deg K	read-only
FWIDELTA	Inner_Filter_OffsetCmd	integer	motor counts	write-only
FWIDEST	Inner_Filter_DestSts	integer	motor counts	read-only
FWIEUP	Inner_Filter_Dest	float	eng units	read-only
FWIHOME	Inner_Filter_Home	boolean	true/false	read/write
FWIIDLE	Inner_Filter_IdleSts	boolean	true/false	read-only
FWINAME	Inner_Filter_Name	string	posn name	read/write
FWIPOS	Inner_Filter_Position	integer	1 of n	read/write
FWIRAW	Inner_Filter_RawPosn	integer	motor counts	read/write
FWISTAT	Inner_Filter_State	string		read-only

**TABLE 20. Mechanism observing keywords.**

FWITRGT	Outer_Filter_TargetPosn	string	posn name	read-only
FWODELTA	Outer_Filter_OffsetCmd	integer	motor counts	write-only
FWODEST	Outer_Filter_DestSts	integer	motor counts	read-only
FWOEUP	Outer_Filter_Dest	float	eng units	read-only
FWOHOME	Outer_Filter_Home	boolean	true/false	read/write
FWOIDLE	Outer_Filter_IdleSts	boolean	true/false	read-only
FWONAME	Outer_Filter_Name	string	posn name	read/write
FWOPOS	Outer_Filter_Position	integer	1 of n	read/write
FWORAW	Outer_Filter_RawPosn	integer	motor counts	read/write
FWOSTAT	Outer_Filter_State	string		read-only
FWOTRGT	Outer_Filter_TargetPosn	string	posn name	read-only
GETTER_T	temperature_diode_9	float	deg K	read-only
GRSDELTA	Grism_OffsetCmd	integer	motor counts	write-only
GRSDEST	Grism_DestSts	integer	motor counts	read-only
GRSEUP	Grism_Dest	float	eng units	read-only
GRSHOME	Grism_Home	boolean	true/false	read/write
GRSIDLE	Grism_IdleSts	boolean	true/false	read-only
GRSNAME	Grism_Name	string	posn name	read/write
GRSPOS	Grism_Position	integer	1 of n	read/write
GRSRAW	Grism_RawPosn	integer	motor counts	read/write
GRSSTAT	Grism_State	string		read-only
GRSTRGT	Grism_TargetPosn	string	posn name	read-only
KRYPTONPWR	Krypton_lamp_ctrl	boolean	true/false	write-only
LAMPPWR	Spectral_lamp_ctrl	boolean	true/false	write-only
LHETEMP	lhe_work_surface_temp	double	deg K	read-only
LNTEMP	liquid_nitrogen_temp	double	deg K	read-only
NEONPWR	Neon_lamp_ctrl	boolean	true/false	write-only
PRESLIT_T	temperature_diode_2	float	deg K	read-only
PRDWDELTA	Pupil_Drive_OffsetCmd	integer	motor counts	write-only
PRDWDEST	Pupil_Drive_DestSts	integer	motor counts	read-only
PRDWEUP	Pupil_Drive_Dest	float	eng units	read-only
PRDWHOME	Pupil_Drive_Home	boolean	true/false	read/write
PRDWIDLE	Pupil_Drive_IdleSts	boolean	true/false	read-only
PRDWNAME	Pupil_Drive_Name	string	posn name	read/write
PRDWPOS	Pupil_Drive_Position	integer	1 of n	read/write

**TABLE 20. Mechanism observing keywords.**

PRDWRW	Pupil_Drive_RawPosn	integer	motor counts	read/write
PRDWSTAT	Pupil_Drive_State	string		read-only
PRDWTRGT	Pupil_Drive_TargetPosn	string	posn name	read-only
PRDWVEL	Pupil_Drive_Velocity	string	posn name	read/write
PSLDELTA	Lower_Preslit_OffsetCmd	integer	motor counts	write-only
PSLDEST	Lower_Preslit_DestSts	integer	motor counts	read-only
PSLEUP	Lower_Preslit_Dest	float	eng units	read-only
PSLHOME	Lower_Preslit_Home	boolean	true/false	read/write
PSLIDLE	Lower_Preslit_IdleSts	boolean	true/false	read-only
PSLNAME	Lower_Preslit_Name	string	posn name	read/write
PSLPOS	Lower_Preslit_Position	integer	1 of n	read/write
PSLRAW	Lower_Preslit_RawPosn	integer	motor counts	read/write
PSLSTAT	Lower_Preslit_State	string		read-only
PSLTRGT	Lower_Preslit_TargetPosn	string	posn name	read-only
PSUDELTA	Upper_Preslit_OffsetCmd	integer	motor counts	write-only
PSUDEST	Upper_Preslit_DestSts	integer	motor counts	read-only
PSUEUP	Upper_Preslit_Dest	float	eng units	read-only
PSUHOME	Upper_Preslit_Home	boolean	true/false	read/write
PSUIDLE	Upper_Preslit_IdleSts	boolean	true/false	read-only
PSUNAME	Upper_Preslit_Name	string	posn name	read/write
PSUPOS	Upper_Preslit_Position	integer	1 of n	read/write
PSURAW	Upper_Preslit_RawPosn	integer	motor counts	read/write
PSUSTAT	Upper_Preslit_State	string		read-only
PSUTRGT	Upper_Preslit_TargetPosn	string	posn name	read-only
PWDELTA	Pupil_Wheel_OffsetCmd	integer	motor counts	write-only
PWDEST	Pupil_Wheel_DestSts	integer	motor counts	read-only
PWEUP	Pupil_Wheel_Dest	float	eng units	read-only
PWHOME	Pupil_Wheel_Home	boolean	true/false	read/write
PWIDLE	Pupil_Wheel_IdleSts	boolean	true/false	read-only
PWNAME	Pupil_Wheel_Name	string	posn name	read/write
PWPOS	Pupil_Wheel_Position	integer	1 of n	read/write
PWRW	Pupil_Wheel_RawPosn	integer	motor counts	read/write
PWSTAT	Pupil_Wheel_State	string		read-only
PWTRGT	Pupil_Wheel_TargetPosn	string	posn name	read-only
SHIELD_T	temperature_diode_2	float	deg K	read-only

**TABLE 20. Mechanism observing keywords.**

SHTRDELTA	Shutter_OffsetCmd	integer	motor counts	write-only
SHTRDEST	Shutter_DestSts	integer	motor counts	read-only
SHTREUP	Shutter_Dest	float	eng units	read-only
SHTRHOME	Shutter_Home	boolean	true/false	read/write
SHTRIDLE	Shutter_IdleSts	boolean	true/false	read-only
SHTRNAME	Shutter_Name	string	posn name	read/write
SHTRPOS	Shutter_Position	integer	1 of n	read/write
SHTRRAW	Shutter_RawPosn	integer	motor counts	read/write
SHTRSTAT	Shutter_State	string		read-only
SHTRTRGT	Shutter_TargetPosn	string	posn name	read-only
SLIT_T	temperature_diode_10	float	deg K	read-only
SLTMDELTA	Slit_Mask_OffsetCmd	integer	motor counts	write-only
SLTMDEST	Slit_Mask_DestSts	integer	motor counts	read-only
SLTMEUP	Slit_Mask_Dest	float	eng units	read-only
SLTMHOME	Slit_Mask_Home	boolean	true/false	read/write
SLTMIDLE	Slit_Mask_IdleSts	boolean	true/false	read-only
SLTMNAME	Slit_Mask_Name	string	posn name	read/write
SLTMPOS	Slit_Mask_Position	integer	1 of n	read/write
SLTMRAW	Slit_Mask_RawPosn	integer	motor counts	read/write
SLTMSTAT	Slit_Mask_State	string		read-only
SLTMTRGT	Slit_Mask_TargetPosn	string	posn name	read-only
SLTSDELTA	Slit_Slide_OffsetCmd	integer	motor counts	write-only
SLTSDEST	Slit_Slide_DestSts	integer	motor counts	read-only
SLTSEUP	Slit_Slide_Dest	float	eng units	read-only
SLTSHOME	Slit_Slide_Home	boolean	true/false	read/write
SLTSIDLE	Slit_Slide_IdleSts	boolean	true/false	read-only
SLTSNAME	Slit_Slide_Name	string	posn name	read/write
SLTSPOS	Slit_Slide_Position	integer	1 of n	read/write
SLTSRAW	Slit_Slide_RawPosn	integer	motor counts	read/write
SLTSSTAT	Slit_Slide_State	string		read-only
SLSTRGT	Slit_Slide_TargetPosn	string	posn name	read-only
TEMPDET	detector_block_temp	double	deg K	read-only
XENONPWR	Xenon_lamp_ctrl	boolean	true/false	write-only

**TABLE 21. Mechanism maintenance keywords.**

NAME	DESCRIPTION	TYPE	UNITS/VALUES	R/W FLAG
CAMBRAKES	Camera_Slide-Brakes_Cmd	boolean	true/false	write-only
CAMENABLE	Camera_Slide_PwrCtrl	boolean	true/false	write-only
CAMINFO	Camera_Slide_Info	boolean	true/false	read-only
CAMKILL	Camera_Slide_KillCmd	boolean	true/false	write-only
CAMRESET	Camera_Slide_ResetCmd	boolean	true/false	write-only
CAMSIM	Camera_Slide_Simulate	boolean	true/false	write-only
CAMSTOP	Camera_Slide_StopCmd	boolean	true/false	write-only
FWIBRAKES	Inner_Filter_BrakesCmd	boolean	true/false	write-only
FWIENABLE	Inner_Filter_PwrCtrl	boolean	true/false	write-only
FWIINFO	Inner_Filter_Info	boolean		read-only
FWIKILL	Inner_Filter_KillCmd	boolean	true/false	write-only
FWIRESET	Inner_Filter_ResetCmd	boolean	true/false	write-only
FWISIM	Inner_Filter_Simulate	boolean	true/false	write-only
FWISTOP	Inner_Filter_StopCmd	boolean	true/false	write-only
FWOBRAKES	Outer_Filter_BrakesCmd	boolean	true/false	write-only
FWOENABLE	Outer_Filter_PwrCtrl	boolean	true/false	write-only
FWOINFO	Outer_Filter_Info	boolean		read-only
FWOKILL	Outer_Filter_KillCmd	boolean	true/false	write-only
FWORESET	Outer_Filter_ResetCmd	boolean	true/false	write-only
FWOSIM	Outer_Filter_Simulate	boolean	true/false	write-only
FWOSTOP	Outer_Filter_StopCmd	boolean	true/false	write-only
GRSBRAKES	Grism_BrakesCmd	boolean	true/false	write-only
GRSEENABLE	Grism_PwrCtrl	boolean	true/false	write-only
GRSINFO	Grism_Info	boolean		read-only
GRSKILL	Grism_KillCmd	boolean	true/false	write-only
GRSRESET	Grism_ResetCmd	boolean	true/false	write-only
GRSSIM	Grism_Simulate	boolean	true/false	write-only
GRSSTOP	Grism_StopCmd	boolean	true/false	write-only
PRDWBRAKES	Pupil_Drive_BrakesCmd	boolean	true/false	write-only
PRDWENABLE	Pupil_Drive_PwrCtrl	boolean	true/false	write-only
PRDWINFO	Pupil_Drive_Info	boolean		read-only
PRDWKILL	Pupil_Drive_KillCmd	boolean	true/false	write-only

**TABLE 21. Mechanism maintenance keywords.**

NAME	DESCRIPTION	TYPE	UNITS/VAL-UES	R/W FLAG
PRDWRESET	Pupil_Drive_ResetCmd	boolean	true/false	write-only
PRDWSIM	Pupil_Drive_Simulate	boolean	true/false	write-only
PRDWSTOP	Pupil_Drive_StopCmd	boolean	true/false	write-only
PSLBRAKES	Lower_Preslit_BrakesCmd	boolean	true/false	write-only
PSLENABLE	Lower_Preslit_PwrCtrl	boolean	true/false	write-only
PSLINFO	Lower_Preslit_Info	boolean		read-only
PSLKILL	Lower_Preslit_KillCmd	boolean	true/false	write-only
PSLRESET	Lower_Preslit_ResetCmd	boolean	true/false	write-only
PSLSIM	Lower_Preslit_Simulate	boolean	true/false	write-only
PSLSTOP	Lower_Preslit_StopCmd	boolean	true/false	write-only
PSUBRAKES	Upper_Preslit_BrakesCmd	boolean	true/false	write-only
PSUENABLE	Upper_Preslit_PwrCtrl	boolean	true/false	write-only
PSUINFO	Upper_Preslit_Info	boolean		read-only
PSUKILL	Upper_Preslit_KillCmd	boolean	true/false	write-only
PSURESET	Upper_Preslit_ResetCmd	boolean	true/false	write-only
PSUSIM	Upper_Preslit_Simulate	boolean	true/false	write-only
PSUSTOP	Upper_Preslit_StopCmd	boolean	true/false	write-only
PWBRAKES	Pupil_Wheel_BrakesCmd	boolean	true/false	write-only
PWENABLE	Pupil_Wheel_PwrCtrl	boolean	true/false	write-only
PWINFO	Pupil_Wheel_Info	boolean		read-only
PWKILL	Pupil_Wheel_KillCmd	boolean	true/false	write-only
PWRESET	Pupil_Wheel_ResetCmd	boolean	true/false	write-only
PWSIM	Pupil_Wheel_Simulate	boolean	true/false	write-only
PWSTOP	Pupil_Wheel_StopCmd	boolean	true/false	write-only
SHTRBRAKES	Shutter_BrakesCmd	boolean	true/false	write-only
SHTRENABLE	Shutter_PwrCtrl	boolean	true/false	write-only
SHTRINFO	Shutter_Info	boolean		read-only
SHTRKILL	Shutter_KillCmd	boolean	true/false	write-only
SHTRRESET	Shutter_ResetCmd	boolean	true/false	write-only
SHTRSIM	Shutter_Simulate	boolean	true/false	write-only
SHTRSTOP	Shutter_StopCmd	boolean	true/false	write-only
SLTMBRAKES	Slit_Mask_BrakesCmd	boolean	true/false	write-only
SLTMENABLE	Slit_Mask_PwrCtrl	boolean	true/false	write-only
SLTMINFO	Slit_Mask_Info	boolean		read-only

**TABLE 21. Mechanism maintenance keywords.**

NAME	DESCRIPTION	TYPE	UNITS/VALUES	R/W FLAG
SLTMKILL	Slit_Mask_KillCmd	boolean	true/false	write-only
SLTMRESET	Slit_Mask_ResetCmd	boolean	true/false	write-only
SLTMSIM	Slit_Mask_Simulate	boolean	true/false	write-only
SLTMSTOP	Slit_Mask_StopCmd	boolean	true/false	write-only
SLTSBRAKES	Slit_Slide_BrakesCmd	boolean	true/false	write-only
SLTSENABLE	Slit_Slide_PwrCtrl	boolean	true/false	write-only
SLTSINFO	Slit_Slide_Info	boolean		read-only
SLTSKILL	Slit_Slide_KillCmd	boolean	true/false	write-only
SLTSRESET	Slit_Slide_ResetCmd	boolean	true/false	write-only
SLTSSIM	Slit_Slide_Simulate	boolean	true/false	write-only
SLTSSTOP	Slit_Slide_StopCmd	boolean	true/false	write-only

**TABLE 22. Motor engineering keywords.**

NAME	DESCRIPTION	TYPE	UNITS/VALUES	R/W FLAG
MTRBRAKESON	Set_motor_brakes	integer	motor number	write-only
MTRBRAKESOFF	Release_motor_brakes	integer	motor number	write-only
MTRRESET	Powerup_reset	integer	motor number	write-only
MTRWAIT	Wait_stopped	integer	motor number	write-only
MTRHOME	Initiate_homing	integer	motor number	write-only
MTRISHOME	Homing_status	integer	motor number	write-only
MTRENABLE	Enable_motor	integer	motor number	write-only
MTRDISABLE	Disable_motor	integer	motor number	write-only
MTRSTOP	Decelerate_to_stop	integer	motor number	write-only
MTRKILL	Abruptly_stop	integer	motor number	write-only
MTRPOSN	Show_motor_position	integer	Motor number	write-only
MTRINFO	Show_motor_info	integer	motor number	write-only
MTR1RAW	motor_1_raw_posn	integer	motor counts	read/write
MTR1REL	motor_1_rel_demand	integer	motor counts	write-only
MTR1HOME	motor_1_home	boolean	true/false	read/write
MTR1IDLE	motor_1_idle_sts	boolean	true/false	read-only
MTR1INFO	motor_1_info	boolean	true_false	write-only

**TABLE 22. Motor engineering keywords.**

<b>NAME</b>	<b>DESCRIPTION</b>			
MTR1TELL	motor_1_tell_cmd	string	Motor language	write-only
MTR1ASK	motor_1_ask_cmd	string	Motor language	write-only
MTR2RAW	motor_2_raw_posn	integer	motor counts	read/write
MTR2REL	motor_2_rel_demand	integer	motor counts	write-only
MTR2HOME	motor_2_home	boolean	true/false	read/write
MTR2IDLE	motor_2_idle_sts	boolean	true/false	read-only
MTR2INFO	motor_2_info	boolean	true_false	write-only
MTR2TELL	motor_2_tell_cmd	string	Motor language	write-only
MTR2ASK	motor_2_ask_cmd	string	Motor language	write-only
MTR3RAW	motor_3_raw_posn	integer	motor counts	read/write
MTR3REL	motor_3_rel_demand	integer	motor counts	write-only
MTR3HOME	motor_3_home	boolean	true/false	read/write
MTR3IDLE	motor_3_idle_sts	boolean	true/false	read-only
MTR3INFO	motor_3_info	boolean	true_false	write-only
MTR3TELL	motor_3_tell_cmd	string	Motor language	write-only
MTR3ASK	motor_3_ask_cmd	string	Motor language	write-only
MTR4RAW	motor_4_raw_posn	integer	motor counts	read/write
MTR4REL	motor_4_rel_demand	integer	motor counts	write-only
MTR4HOME	motor_4_home	boolean	true/false	read/write
MTR4IDLE	motor_4_idle_sts	boolean	true/false	read-only
MTR4INFO	motor_4_info	boolean	true_false	write-only
MTR4TELL	motor_4_tell_cmd	string	Motor language	write-only
MTR4ASK	motor_4_ask_cmd	string	Motor language	write-only
MTR5RAW	motor_5_raw_posn	integer	motor counts	read/write
MTR5REL	motor_5_rel_demand	integer	motor counts	write-only
MTR5HOME	motor_5_home	boolean	true/false	read/write
MTR5IDLE	motor_5_idle_sts	boolean	true/false	read-only
MTR5INFO	motor_5_info	boolean	true_false	write-only
MTR5TELL	motor_5_tell_cmd	string	Motor language	write-only
MTR5ASK	motor_5_ask_cmd	string	Motor language	write-only
MTR6RAW	motor_6_raw_posn	integer	motor counts	read/write
MTR6REL	motor_6_rel_demand	integer	motor counts	write-only
MTR6HOME	motor_6_home	boolean	true/false	read/write
MTR6IDLE	motor_6_idle_sts	boolean	true/false	read-only

**TABLE 22. Motor engineering keywords.**

NAME	DESCRIPTION			
MTR6INFO	motor_6_info	boolean	true_false	write-only
MTR6TELL	motor_6_tell_cmd	string	Motor language	write-only
MTR6ASK	motor_6_ask_cmd	string	Motor language	write-only
MTR7RAW	motor_7_raw_posn	integer	motor counts	read/write
MTR7REL	motor_7_rel_demand	integer	motor counts	write-only
MTR7HOME	motor_7_home	boolean	true/false	read/write
MTR7IDLE	motor_7_idle_sts	boolean	true/false	read-only
MTR7INFO	motor_7_info	boolean	true_false	write-only
MTR7TELL	motor_7_tell_cmd	string	Motor language	write-only
MTR7ASK	motor_7_ask_cmd	string	Motor language	write-only
MTR8RAW	motor_8_raw_posn	integer	motor counts	read/write
MTR8REL	motor_8_rel_demand	integer	motor counts	write-only
MTR8HOME	motor_8_home	boolean	true/false	read/write
MTR8IDLE	motor_8_idle_sts	boolean	true/false	read-only
MTR8INFO	motor_8_info	boolean	true_false	write-only
MTR8TELL	motor_8_tell_cmd	string	Motor language	write-only
MTR8ASK	motor_8_ask_cmd	string	Motor language	write-only
MTR9RAW	motor_9_raw_posn	integer	motor counts	read/write
MTR9REL	motor_9_rel_demand	integer	motor counts	write-only
MTR9HOME	motor_9_home	boolean	true/false	read/write
MTR9IDLE	motor_9_idle_sts	boolean	true/false	read-only
MTR9INFO	motor_9_info	boolean	true_false	write-only
MTR9TELL	motor_9_tell_cmd	string	Motor language	write-only
MTR9ASK	motor_9_ask_cmd	string	Motor language	write-only
MTR10RAW	motor_10_raw_posn	integer	motor counts	read/write
MTR10REL	motor_10_rel_ demand	integer	motor counts	write-only
MTR10HOME	motor_10_home	boolean	true/false	read/write
MTR10IDLE	motor_10_idle_sts	boolean	true/false	read-only
MTR10INFO	motor_10_info	boolean	true_false	write-only
MTR10TELL	motor_10_tell_cmd	string	Motor language	write-only
MTR10ASK	motor_10_ask_cmd	string	Motor language	write-only
MTR11RAW	motor_11_raw_posn	integer	motor counts	read/write
MTR11REL	motor_11_rel_ demand	integer	motor counts	write-only

**TABLE 22. Motor engineering keywords.**

<b>NAME</b>	<b>DESCRIPTION</b>			
TR11HOME	motor_11_home	boolean	true/false	read/write
MTR11IDLE	motor_11_idle_sts	boolean	true/false	read-only
MTR11INFO	motor_11_info	boolean	true_false	write-only
MTR11TELL	motor_11_tell_cmd	string	Motor language	write-only
MTR11ASK	motor_11_ask_cmd	string	Motor language	write-only

## Appendix D. KTL Keywords

The KTL/keyword standard has been used at Keck since it was first introduced in 1991. The documents that describe the KTL/keyword protocol are

- 1991 Keck Software Document “KTL: The Keck Task Library” (KSD-8)
- 1993 ADASS paper “The Keck Keyword Library” (A. R. Conrad and W. F. Lupton, “The Keck Keyword Layer,” *Astronomical Data Analysis Software and Systems II*, Astronomical Society of the Pacific Conference Series, Vol. 52, pp. 203-212).

### D.1 Figures

Figures 6 and 7 depict the goals of the KTL Keyword Library, a common API and plug compatible applications.

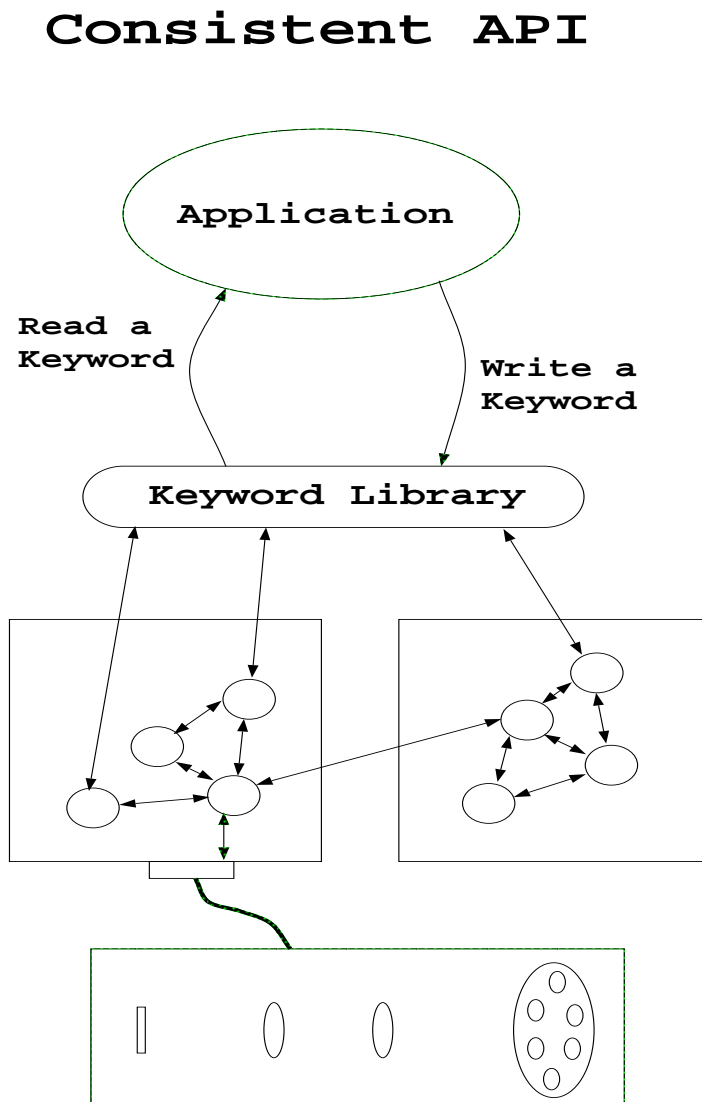


FIGURE 8. Consistent API.

# Plug Compatible Applications

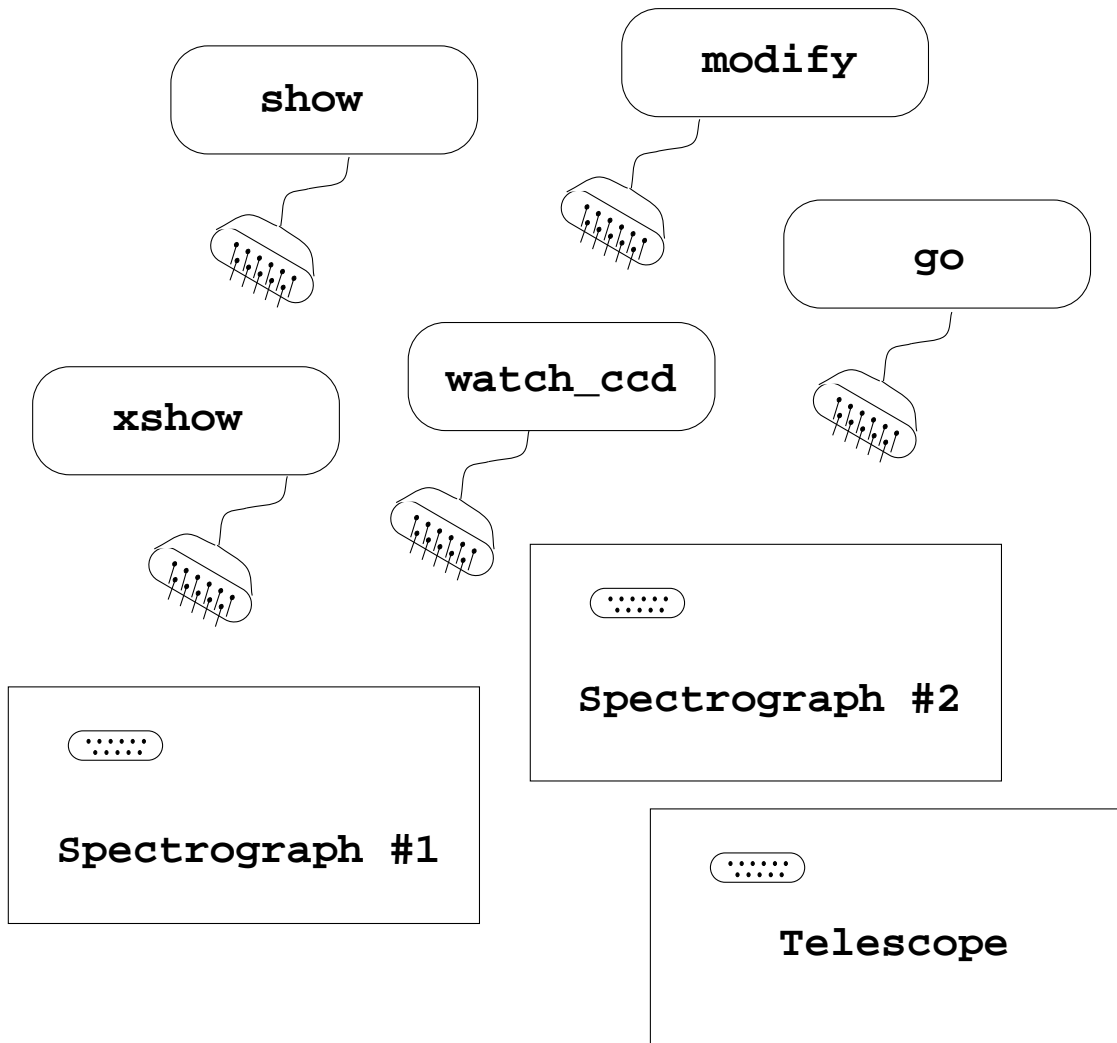


FIGURE 9. Plug compatible applications.

## D.2 Shell Level Commands

**TABLE 23. Shell level commands.**

<code>show -s <i>subsystem</i> KEYWORD</code>	Reads the current value of a keyword
<code>show -2 <i>subsystem</i> keywords</code>	Displays a list of all available keywords for this subsystem
<code>show -s <i>subsystem</i> KEYWORD=VALUE [ WAIT/NOWAIT]</code>	Sets a KEYWORD to indicated VALUE, optionally waiting for completion of operation
<code>waitfor -s <i>subsystem</i> KEYWORD=VALUE</code>	Waits for KEYWORD to achieve VALUE
<code>xshow -s <i>subsystem</i> KEYWORD</code>	Provides continuous display of KEYWORD

## D.3 C Calling Sequences

**TABLE 24. C calling sequences.**

<code>kctl_open (subsystem, .. , handle)</code>	Returns handle to given subsystem
<code>kctl_read ( handle, ll , keyword, &amp;value )</code>	Reads the current value of a keyword
<code>kctl_write ( handle, .. , keyword, value, [WAIT   NOWAIT] )</code>	Sets a keyword to a value, optionally waiting for completing of operation
<code>kctl_read ( handle, KCTL_CONTINUOUS, keyword, callback)</code>	Sets up a continuous read on a keyword. Callback is invoked whenever keyword changes value.
<code>kctl_ioctl(handle, KCTL_HEADER, keywords, values, comments)</code>	Read current values of a keyword block suitable for FITS header.

## D.4 Example Commands

**TABLE 25. Shell level commands**

<code>show -s hires tempdet</code>	Show current value of HIRES detector temperature
<code>show -s lriss tempdet</code>	Show current value of LRIS detector temperature
<code>show -s hires keywords   fgrep LAMP</code>	Displays all HIRES keywords related to lamps

**TABLE 25. Shell level commands**

---

modify -s hires slitwid=10.0	Set HIRES slit width to 10.0 arc-seconds, and wait for the motion to complete
modify -s hires camcover=open nowait	Request the camera cover to open, but don't wait for motion to complete
waitfor -s dcs axestat=tracking	Wait for the last telescope offset to complete
xshow -s hires relhum	Provide continuous display of relative humidity

## Appendix E. Glossary

ALADDIN - The type of detector being used in both NIRC2 and NIRSPEC.

API - Application Programmers Interface.

Animatics - A commercial provider of motor controllers ([www.animatics.com](http://www.animatics.com)).

CARA - California Association for Research in Astronomy.

CLI - Command Line Interface.

DCS - Drive and Control System. The telescope control software (often called TCS at other observatories).

DGH - A commercial provider of data acquisition electronics ([www.dghcorp.com](http://www.dghcorp.com)).

FITS - Flexible Image Transport System. Image format standard used in astronomy.

GUI - Graphical User Interface.

IDL - Interactive Data Language. A commercial math and plotting package.

IRAF - Image Reduction and Analysis Facility. A public domain image reduction package designed specifically for astronomy.

KTL - Keck Tasking Library. A keyword-based protocol that defines a common command language and API at Keck (Appendix D).

Lakeshore - A commercial provider of temperature controllers ([www.lakeshore.com](http://www.lakeshore.com)).

NIRC or NIRC1 - The first generation near infrared camera for the Keck telescope.

NIRC2 - The second generation near infrared camera for the Keck telescope.

NIRSPEC - Near Infrared Echelle Spectrograph. An instrument similar to NIRC2 being provided by UCLA.

RPC - Remote Procedure Call. A software standard for interprocess communication.

RSI - Research Systems, Inc. Producers of IDL.

UCLA - The University of California at Los Angeles.

**Appendix F. List of Figures**

NIRC2 Software Overview.....	6
Screen Dump of a NIRC1 Session.....	15
Status Display.....	16
Pupil Assembly.....	46
NIRCSPEC Software.....	47
Schedule .....	56
Workload.....	57
Consistent API.....	79
Plug compatible applications.....	80

## Appendix G. List of Tables

TABLE 1.	NIRC2 shell scripts and their functions. ....	7
TABLE 2.	Optical bench mechanisms. ....	17
TABLE 3.	Prefixes to mechanisms' associations. ....	18
TABLE 4.	Keyword suffixes and their descriptions. ....	19
TABLE 5.	Pupil rotation drive unique keywords. ....	21
TABLE 6.	Thermal and spectral lamp keywords. ....	21
TABLE 7.	Maintenance keyword suffixes. ....	22
TABLE 8.	General purpose motor keywords. ....	23
TABLE 9.	Motor-specific keywords. ....	24
TABLE 10.	Fields of the keyword configuration files. ....	26
TABLE 11.	Fields of the mechanism config file. ....	27
TABLE 12.	Simulation environment variables. ....	30
TABLE 13.	Motor information for specification. ....	33
TABLE 14.	Motor characteristics. ....	34
TABLE 15.	Motor daemon support programs. ....	35
TABLE 16.	Information for device specification. ....	42
TABLE 17.	Device characteristics. ....	43
TABLE 18.	I/O daemon support programs. ....	44
TABLE 19.	Requirements cross reference. ....	60
TABLE 20.	Mechanism observing keywords. ....	69
TABLE 21.	Mechanism maintenance keywords. ....	73
TABLE 22.	Motor engineering keywords. ....	75
TABLE 23.	Shell level commands. ....	81
TABLE 24.	C calling sequences. ....	81
TABLE 25.	Shell level commands. ....	81