4444444444444444444444444444444444444444444444444444444444444444444444444444444444

NIRSPEC

UCLA Astrophysics Program **U.C. Berkeley** **W.M.Keck Observatory**

4444444444444444444444444444444444444444444444444444444444444444444444444444444444

Tim Liu

May 3, 1996

NIRSPEC Software Design Note 6.00
Keyword Library

1 Introduction

Similar to existing Keck instruments, the NIRSPEC software will provide a keyword library that interfaces between the KTL (Keck Task Library) based user interface applications and the underlying hardware. This will provide a consistent programming interface with other Keck software for writing KTL applications. In fact, the implementation of a keyword library is required by the Keck Software Coordination Committee (SCC). This document outlines the design and initial implementation of the NIRSPEC keyword library. A preliminary list of NIRSPEC keywords is also presented.

2 Keyword Interface Layer

Figure 1 shows how a keyword layer interfaces to other subsystems. In a keyword style control flow approach, user inputs for application control are sent to the underlying hardware control applications via *keywords*. Similarly, application data (status info, not pixel data) are also returned to the KTL based user interface program in *keywords*.

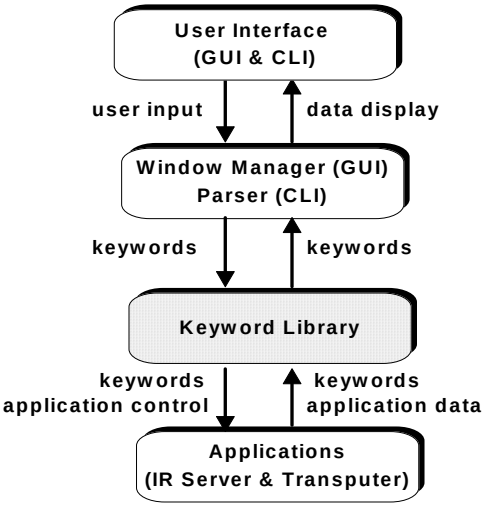

Figure 1 Keyword Interface Layer

The main advantage of such a keyword approach to the instrument control software is a consistent API (Application Programming Interface). When one writes an application on top of a Keck keyword library, he/she will always use the same half dozen KTL functions like *ktl_open(), ktl_read(), ktl_write(),* etc., regardless of which hardware system the keyword library controls. The keyword layer spares the applications programmer from having to deal with the complexities of the hardware directly.

3 Message Systems

A keyword library communicates with the underlying subsystems via some inter-process communication mechanism, or message system. At present, there're three message systems used by the Keck telescope and instrument software: RPC (Remote Procedure Call), MUSIC (Multi User System for Instrument Control), and CA (Channel Access). There's no SCC mandate on which system should be used for Keck II keyword libraries. It appears that the telescope control software systems are converging to CA. On the other hand, none of the Keck II instruments have committed to the use of CA. In fact, it is almost certain that DEIMOS/ESI and NIRC2 will not use CA. One of the major benefits from using CA, in addition to various client tools such as alarm handler, archiver, display manager, etc., is the high bandwidth capability for handling message flow between the server and clients. For instrument software, however, the traffic between the keyword library and the hardware is much lighter compared to the telescope control systems, and RPC and MUSIC are sufficient to handle it.

The major problem for us to use CA is the lack of technical support and documentation. The current CA (actually called portable channel access which is running under Unix, to differentiate from the EPICS version which requires VxWorks) is still a beta release (its kernel is being re-coded in C++) and no programming manuals are available. Unlike CARA which has received contracted support and training classes from Los Alamos where CA has been developed, we would certainly face a steep learning curve if we decide to use it, even though we can get help from CA experts at CARA like William Lupton. For MUSIC which is used by a very small community (Lick and Keck I), the main concern is its future support. The lack of detailed programming documentation is also a concern, though not serious because we can copy some of the HIRES code which uses MUSIC.

In contrast, RPC doesn't have these problems because it is the Unix standard for inter-process communications and network programming (it was originally developed by Sun and is bundled with Solaris). It will almost be guaranteed for some longevity. Both LWS and LWIRC use RPC and they're happy about it. Furthermore, we have experience with RPC from the prototyping of the "one-shot" read/write functions in the NIRSPEC keyword library. The various RPC based keyword functions seem to perform well for our prototype. Another advantage with RPC is that, unlike MUSIC, the keyword knowledge is maintained at the server side, which makes it easier to implement the software system under a client-server architecture for multi-session remote observing.

Based on these considerations and also consultation with Al Conrad from CARA, we decide to use RPC to implement both the continuous keyword read (also called “broadcast” or “monitor”) and the "one-shot" blocking read/write functions in the NIRSPEC keyword library.

4 Implementation

Figure 2 illustrates the NIRSPEC keyword interface structure implemented in RPC. The KTL clients GUI and CLI (Command Line Interface) access to the keyword library using the standard KTL function calls. The keyword library is built on top of a RPC client which communicates with an independent RPC server program.The RPC server talks to the transputer data acquisition process via the TSP (Transtech SCSI Processor) link interface.

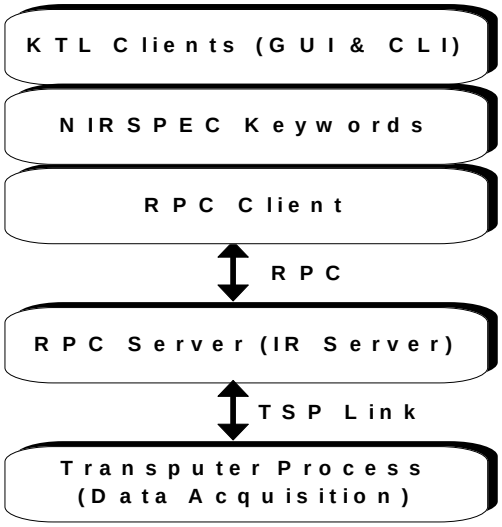

Figure 2 Keyword interface structure

Based on this structure, a prototype NIRSPEC keyword library has been developed for the “one-shot” keyword read/write functions. The continuous read is to be implemented. The source code is located in the directory */kroot/kss/nirspec*.The major source modules and their brief descriptions are listed below:

- *nirspec_keyword.c:* keyword library routines
- *kw_interface.c:* interface between keywords and RPC remote procedures
- *kw_svc_procs.c:* RPC remote procedures
- *kw_svc.c:* RPC server stub file
- *kw_clnt.c:* RPC client stub file
- *kw_xdr.c:* RPC data format exchange file
- *client.c:* RPC client test program

In addition, the directory also contains a library for host-to-transputer communication:

! *tspcom.c:* routines to handle host-transputer link I/O

The KTL keyword service is provided by *nirspec_keyword.c* which includes the following routines: *keyword_init(), keyword_open(), keyword_ioctl(), keyword_read(), keyword_write(), keyword_close(), keyword_event(),* and *keyword_respond()*. These keyword service routines call RPC functions in other source modules to perform the actual hardware tasks.

The make file in the */kroot/kss/nirspec* directory will generate a shareable library called *libnirspec_keyword.so.0.0* which is linked to the NIRSPEC program. The make file will also produce a RPC server executable: *nirspec_rpc_server*. The RPC server program is started by the *keyword_open()* routine when the keyword library is first opened and the server process is closed by *keyword_open()*.

5 NIRSPEC Keyword List

A preliminary list of keywords has been defined for NIRSPEC. They are grouped into several categories as shown below. Each keyword line follows the format:

keyword type function

The keywords have four different data types, including boolean (B), integer (I), double (D), and string (S).

General Setup keywords:

observer	S	specify observers' names
instname	S	specify instrument name
datadir_s	S	specify image data directory (spectrometer)
rootname_s	S	specify image file root name (spectrometer)
filenum_s	I	specify image file running number (spectrometer)
datadir_c	S	specify image data directory (slitview camera)
rootname_c	S	specify image file root name (slitview camera)
filenum_c	I	specify image file running number (slitview camera)
obsmode	S	specify observing mode (stare, node, etc.)
engpasswd	S	specify engineering password to access to engineering functions

Observing Parameter Keywords:

filename_s	S	specify image file name (spectrometer)
object_s	S	specify object name (spectrometer)
comment_s	S	specify comment line (spectrometer)
itime_s	D	specify integration time per coadd in seconds (spectrometer)
coadds_s	I	specify number of coadds (spectrometer)
sampmode_s	I	specify sampling mode (spectrometer)
nreads_s	I	specify number of multiple reads (spectrometer)

filename_c	S	specify image file name (slitview camera)
object_c	S	specify object name (slitview camera)
comment_c	S	specify comment line (slitview camera)
itime_c	D	specify integration time per coadd in seconds (slitview camera)
coadds_c	I	specify number of coadds (slitview camera)
sampmode_c	I	specify sampling mode (slitview camera)
nreads_c	I	specify number of multiple reads (slitview camera)

Exposure Control Keywords:

go_s	B	start an exposure (spectrometer)
abort_s	B	abort current exposure (spectrometer)
framerdy_s	B	signal end of exposure by transputer (spectrometer)
ttime_s	D	give total integration time (= itime*coadds) (spectrometer)
elaptime_s	D	give elapsed total integration time in seconds (spectrometer)
timeleft_s	I	give total remaining integration time in seconds (spectrometer)
go_c	B	start an exposure (slitview camera)
abort_c	B	abort current exposure (slitview camera)
framerdy_c	B	signal end of exposure by transputer (slitview camera)
ttime_c	D	give total integration time (= itime*coadds) (slitview camera)
elaptime_c	D	give elapsed total integration time in seconds (slitview camera)
timeleft_c	I	give total remaining integration time in seconds (slitview camera)

Other Observing Related Keywords:

obsdate_s	S	give starting local date of an exposure (spectrometer)
obstime_s	S	give starting local time of an exposure (spectrometer)
obsdate_c	S	give starting local date of an exposure (slitview camera)
obstime_c	S	give starting local time of an exposure (slitview camera)

Motion Control Keywords:

lampname	S	select named lamp
mirhome	B	move calibration lamp mirror to home position (out)
mirname	S	move lamp mirror to named discrete position (in or out)
mirmove	I	move lamp mirror in steps
mirpos	I	read lamp mirror current position in steps
pinhome	B	move calibration lamp pinhole to home position (out)
pinname	S	move lamp pinhome to named discrete position (in or out)
pinmove	I	move lamp pinhome in steps
pinpos	I	read lamp pinhole current position in steps
rothome	B	move image rotator to home position
rotangle	D	move image rotator in degrees
rotmove	I	move image rotator in steps
rotpos	I	read image rotator current position in steps

filhome	B	move filter wheel to home position (blank)
filname	S	move filter wheel to named discrete position (12 positions)
filmove	I	move filter wheel in steps
filpos	I	read filter wheel current position in steps
slithome	B	move slit wheel to home position
slitname	S	move slit wheel to named discrete position (12 positions)
slitmove	I	move slit wheel in steps
slitpos	I	read slit wheel current position in steps
echhome	B	move echelle selector to home position
echname	S	move echelle selector to named discrete position (3 positions)
echmove	I	move echelle selector in steps
echpos	I	read echelle selector current position in steps
egrahome	B	move echelle grating scanner to home position
egraname	S	move echelle grating scanner to named positions (10 positions)
egramove	I	move echelle grating scanner in steps
egrapos	I	read echelle grating scanner current position in steps
xgrahome	B	move cross-dispersion grating scanner to home position
xgraname	S	move cross-dispersion grating scanner to named positions (10 pos)
xgramove	I	move cross-dispersion grating scanner in steps
xgrapos	I	read cross-dispersion grating scanner current position in steps

Housekeeping Function Keywords:

tempdet_s	D	give detector temperature (spectrometer)
tempdet_c	D	give detector temperature (slitview camera)
tempsens[1-n]	D	give temperature reading of sensor[1-n]
tempalrm	B	indicate state of temeprature alarm switch in electronics cabinet
ln2level	D	indicate LN2 can level (?)

Host-Transputer Communication Keywords:

linkreset	B	reset TSP link
linktrace	B	turn on/off TSP link trace switch
linkread	S	read a data packet from TSP link
linkwrite	S	write a data packet to TSP link

DCS Control Keywords:

nodtrow	D	specify nodding throw in arcsecs
nodangle	D	specify nodding position angle in degrees

Data I/O Keywords:

bitpix_s	I	specify bits per pixel for image file (spectrometer)
savetest_s	B	save test frame (spectrometer)
bitpix_c	I	specify bits per pixel for image file (slitview camera)
savetes_c	B	save test frame (slitview camera)
overwrite	B	turn on/off image file overwrite flag

Data Reduction Pipeline Keywords:

redpipe	B	turn on/off data reduction pipeline

6 Actions

The following actions are to be taken over the next few months:

- implement continuous keyword read function and test
- define more keywords as hardware and software development goes on
- continue to interact with CARA on the standardization of keywords