NIRSPEC

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

Tim Liu **May 15, 1997**
Modified by James Larkin **March 20, 1999**

NIRSPEC Software Programming Note 02.01
Graphical User Interface

1 Introduction

This programming document describes the implementation of the NIRSPEC Graphical User Interface (GUI). The GUI is a DataViews-based X application. A quick guide to learning DataViews is included in the GUI design note NSDN0501.

2 Overview

All the GUI source code is located in the directory **`/kroot/kui/xnirspec`**. In addition, the DataViews **`.v`** view files are located in **`/kroot/kui/xnirspec/views`**, the **`.lay`** layout files in **`/kroot/kui/xnirspec/layouts`**, and **`.dr`** drawing files in **`/kroot/kui/xnirspec/drawings`**.

The GUI source modules are listed as follows:

`xnirspec.c`	- main program
`dataviews.c`	- routines to handle X display using DataViews
`create_interest.c`	- routines to create keyword interest for broadcast
`callbacks.c`	- callbacks for keyword broadcast
`misc.c`	- miscellaneous routines
`xnirspec.h`	- NIRSPEC client definitions
`dataviews.h`	- DataViews definitions
`nirspec.h`	- NIRSPEC server definitions

3 Program Description

3.1 header files

The header file **xnirspec.h** defines various macros and declares program variables as memory buffers to hold the DataViews variable descriptors. Several data structures are also defined, along with public function prototyping.

The header file **dataviews.h** defines data structures for DataViews displays, menus and pop-ups.

3.2 xnirspec.c:

The module **xnirspec.c** contains **main()** of the GUI program. Besides several standard header files, it includes **"xnirspec.h"** and the generic KTL definition header file **"ktl.h"**. A few global variables are declared which can be accessed by other source modules as external variables:

```
int  Simulate = FALSE;          /* simulation mode flag                    */
int  NoDCS = FALSE;             /* no DCS server running flag              */
int  Quit = FALSE;              /* program quit flag                       */
int  ServerOnHost = FALSE;      /* flag indicating server is on the host   */
KTL_HANDLE *khand;              /* handle to the NIRSPEC keyword library   */
KTL_HANDLE *dcs_khand;          /* handle to the DCS keyword library       */
```

main() has the following program control flow and structure:

- Check arguments
- Disable Ctrl-C
- Check whether client and server on the same machine
- Set up error logging
- Make connection to NIRSPEC keyword library
- Create keyword broadcast interest list
- Make connection to DCS keyword library if specified (normally not).
- Initialize global variables
- Initialize X displays
- Loop to process X, and KTL events
    - Create a fd set consisting of X socket and KTL fd
    - Block until an X, or KTL event arrives
    - Handle events
- Close X displays
- Close connection to DCS keyword library
- Close connection to NIRSPEC keyword library

Some explanations are given below:

1. The function **main()** accepts up to three command line arguments. They can be "**-s**", "**-nodcs**" and "**-noql**".

2. Ctrl-C is disabled to prevent abnormal exit of the program.

3. The flag **ServerOnHost** indicates whether the client is running on the same machine as the NIRSPEC server program. If it's **FALSE**, a remote data transfer may be required for the quick-look display.

4. The message logging is performed by the Unix **syslog()** function.

5. Depending upon the value of the flag **Simulate** which is set by the argument "**-s**" in **main()**, the program connects to either the real instrument server or the simulation mode server.

6. The function **init_globalvars()** in **misc.c** is called to set various program flags and initialize variables by reading keyword values from the server.

7. The CLI program is no longer run.

8. **DV_init()** from **dataviews.c** initialize DataViews displays. The X socket file descriptor **x_fd** is returned from the macro **ConnectionNumber()** for the asynchronous I/O in the event loop.

9. A socket connection is opened between the GUI and quick-look to pass information like frame arrival.

10. The core of **main()** is the event loop to process events from the KTL RPC server and events from the X socket, CLI socket and quick-look socket. Because the program must be able to handle several different file I/O sources, an asynchronous I/O multiplexing scheme is implemented for the event loop using the UNIX **select()** function call. **select()** examines an I/O file descriptor sets to see if any of the file descriptors are ready for reading, writing, or have an exceptional condition. A fd set consisting of various fds is created in the beginning of the loop as follows:

```
FD_ZERO( &readfds );
ktl_ioctl( khand, KTL_FDSET, &readfds );
FD_SET( x_fd, &readfds );
FD_SET( cli_fd, &readfds );
FD_SET( ql_fd, &readfds );
```

The macro **FD_ZERO()** initializes a file descriptor set to the null set. Note that because the KTL call **ktl_ioctl(,KTL_FDSET,,)** automatically clears a fd set, it must be placed before the macro **FD_SET()**.

The next code segment in the event loop is to block the process indefinitely until an event arrives:

```
if ( (select( maxfds, &readfds, NULL, NULL, NULL ) == -1 ) &&
      errno != EINTR) ) {
    syslog( LOG_WARNING, "select() failed." );
}
else {
    if ( FD_ISSET( x_fd, &readfds )   ||
         FD_ISSET( ql_fd, &readfds ) ) {
        /*
         *  Handle X events
         */
        if ( FD_ISSET( x_fd, &readfds ) )
            DV_handle();
}
    /*
     *  Invoke KTL event handler
     */
    else
        KTL_DISPATCH( khand );
}
```

select() returns if any of the fds is ready for reading. The program calls the macro **FD_ISSET()** to determine which fd is ready, and then invokes a specific function routine to perform the request operation.

11. When main() exits, RPC and socket connections are closed by clean-up routines.

3.3 dataviews.c:

The routines in **dataviews.c** handle DataViews displays. They can be divided into different groups in terms of functions they perform as listed below:

<u>High-level routines:</u>

DV_init() - initialize DataViews
DV_handle() - handle DataViews events
DV_close() - close DataViews displays
DV_updateScreen() - update display screen
DV_updateCurrentObs() - update current observing parameters
DV_expStatus() - update exposure status display

Routines to perform initialization:

create_screen() - create a window and load the view to be displayed
drawport_init() - load the view, get the objects and the variables, and create a drawport to be used
drawport_new() - load a view, create a new drawport and display.
vdps_init() - initialize variable descriptors
vdps_rebind() - modify the variable descriptor to use our own program variable as the memory buffer
input_objects_init() - initialize input object components and post service result request

Routines to display pop-ups:

popup_draw() - add a popup to the active drawport's view
popup_delete() - delete popup objects from the view
popup_deleteAll() - delete all popup objects from the view

Routines to display sub-views:

subview_load() - load the dialog view, get the interesting objects and create a drawport
subview_draw() - draw subview and handle events
subview_erase() - Erase the subview and repair any damage to other views caused by the erasure

Routines to display dialog boxes:

get_automsg() - display message
get_message() - display a single line message
get_mmessage() - display multiple line message
get_confirm() - get confirmation
get_input() - get an input value

DataViews event callback routines:

menu_inst_setup_callback()
.....

The high-level routines provide a DataViews interface layer to NIRSPEC application routines. The routines **`DV_init()`**, **`DV_handle()`** and **`DV_close()`** are briefly discussed below:

`Display *DV_init( void ):`

This initialization routine performs the following functions:

- Initialize DV-Tools
- Initialize the real-time name:data look up table
- Make entries for the name buffer pairs defined in data_table
- Initialize variable descriptors
- Initialize display
- Initialize input objects
- Initialize all dialog subviews
- Extract environment variable values
- Get exposure status objects for exposure update

`void DV_handle( void ):`

This routine consists of an event loop to handle various X events as shown below:

```
/*
 *  Poll event queue for locator events
 */
while ( location = VOloWinEventPoll( V_NO_WAIT ) ) {
  current_drawport = TloGetSelectedDrawport( location );
  current_screen   = VOloScreen( location );

  VUerHandleLocEvent( location );

  /*
   *  Return event types
   */
  switch ( VOloType( location ) ) {
      case V_RESIZE:
        TscReset( current_screen );
          break;

      case V_EXPOSE:
          TscRedraw( current_screen, VOloRegion( location ));
          break;

    case V_KEYPRESS:
        break;

      case V_BUTTONPRESS:
          handle_button_press( location );
          break;
```

```
        case V_BUTTONRELEASE:
            break;

          default:
              break;
      }
```

For example, a button press will invoke **`handle_button_press()`** to perform certain function. The loop updates observing setup parameters by calling **`update_obsparam()`**. The graphic displays are updated by **`TdpDrawNext()`** at the end of each loop.

`void DV_close( void ):`

This routine is rather simple. It destroys each view and drawport and closes each screen. It then frees the data table of "name:buffer" pairs. That source code is listed below:

```
for ( i = 0; i < NUM_WINS; i++ ) {
    TdpDestroy( dv_drawport[i] );
    TviDestroy( dv_view[i] );
    TscClose( dv_screen[i] );
}

table = data_symbol_table;
while ( VTstlen( table ) > 0 ) {
    node = VTstsnget( table, 0 );
    varname = VTsnkey( node );
    S_FREE( varname );
    VTstsnremove( table, node );
}
VTstdestroy( data_symbol_table );

TTerminate();
```

3.4 create_interest.c and callbacks.c:

When the GUI receives a keyword sent from the NIRSPEC server via broadcast, a user-defined callback function will be invoked by **`KTL_DISPATCH()`** to perform certain function. Whether the program should respond to a NIRSPEC keyword broadcast from the server is defined by **`create_nirspec_interest()`** in **`create_interest.c`** using the defined macro **`EXPRESS_INTEREST`**:

```
#define EXPRESS_INTEREST( keyword, keyword_callback )                    \
  context.callback = (int(*)()) keyword_callback;                        \
  if (ktl_read(khand,KTL_CONTINUOUS|KTL_NOPRIME,keyword,0,0,&context)<0) { \
      fprintf( stderr, "xnirspec: %s\n", ktl_get_errtxt() );             \
      exit( -1 );                                                        \
  }
```

For example, **`EXPRESS_INTEREST("outdir",outdir_callback)`** will allow the broadcast of the keyword "**`outdir`**" to invoke a callback function.

A callback routine performs a user-defined function when a keyword broadcast happens. The follow is the callback function for the keyword "**`outdir`**":

```
void outdir_callback( keyword, user_data, call_data, context )
char *keyword;                      /* keyword                          */
void *user_data;                    /* unused                           */
KTL_POLYMORPH *call_data;           /* contains new value               */
KTL_CONTEXT *context;               /* command context (unused)         */
{
    strcpy( VdpBuf_datapath[0], call_data->s );
    strcpy( ObsSetup[0].datapath, call_data->s );
    DV_updateScreen( SCREEN_SPEC );
}
```

As can be seen, this callback copies the new **`outdir`** value to a display variable and updates the DataViews display.

3.5 misc.c:

`misc.c` contains miscellaneous functions which include routines to control exposures, routines to make a file name, routines to manipulate instrument configuration file, and other routines. They are listed below:

`exp_start()`	- start an exposure
`exp_abort()`	- abort an exposure
`exp_end()`	- finish an exposure
`exp_done()`	- finish an exposure and notify quick-look display
`exp_updateStatus()`	- update exposure status
`write_obsParam()`	- write observing parameters using **`ktl_write()`**
`write_abort()`	- write the "abort" keyword
`Filename_get()`	- get the current image file name
`Filename_make()`	- make an image file name by incrementing file number
`config_save()`	- save instrument configuration file
`config_read()`	- read configuration file and re-configure instrument
`init_globalvars()`	- initialize global variables
`get_hostname()`	- get the host domain name