Tim Liu **March 7, 1997**

NIRSPEC Software Programming Note 03.00
Quick Look External Interface

1 Introduction

The **NIRSPEC** quick look (**QL**) facility is programmed in IDL widgets, while the other parts of the **NIRSPEC** high-level software system are coded in C. The **QL** must be able to communicate with the **GUI** to pass messages. In order to interface the two different software sub-systems, a communication scheme has been developed in the design note NSDN1200. This programming document describes the implementation of the **QL** external interface design. In addition, program building and execution are also discussed. As usual, the reader should consult the design note NSDN1200 before reading this programming note.

2 Overview

As discussed in the design note, the **QL**-**GUI** interface uses a UNIX socket and the IDL CALL_EXTERNAL call mechanism to carry out a communication. The software interface consist of two parts, a socket server and a socket client. The socket server contains interface routines for the **GUI**, while the client software provides socket I/O routines that are called by IDL programs using CALL_EXTERNAL.

The **QL** external interface routines are in the two source modules **ql_server.c** and **ql_client.c**. In addition, the low level socket functions used by the two source files are contained in **socket.c**. This note will only describe routines in **ql_server.c** and **ql_client.c**, because **socket.c** has been covered by NSPN0200 (Programming Note on Command Line User Interface). No special user header files are used by the source modules except for standard UNIX include files. We list these source modules below:

ql_server.c	- socket server routines used by the **GUI**
ql_client.c	- socket client routines invoked by IDL CALL_EXTERNAL calls
socket.c	- low-level socket routines

These source modules along with the make file and compiled objects are located in the **NIRSPEC** client software directory **/kroot/kui/xnirspec**.

3 Program Structure

3.1 ql_server.c

The module **ql_server.c** contains the following three function routines that can be called from the **GUI** program:

```
int  QL_serverOpen( int timeout )          - open a socket connection to QL
int  QL_serverIO( int rw, char *msg )      - perform a socket I/O
void QL_serverClose( void)                 - close socket connection
```

1. When the **GUI** program starts, it will call **QL_serverOpen()** to open a socket connection to the **QL**. If the socket fails to open in **timeout** seconds, the routine will return **-1**, indicating a failure.

This server opening routine first gets the socket directory path from the environment variable **NIRSPEC_SOCKET_DIR** which is defined in the **NIRSPEC** client software initialization file **NirspecClientInit** located in the same directory:

```
if ( (dir = (char *)getenv( "NIRSPEC_SOCKET_DIR" )) == NULL )
    return -1;
sprintf( socket_name, "%s/nirspec_ql_socket0", dir );
```

The routine then calls a socket opening routine in **socket.c** as follows:

```
if ( socket_serverOpen( socket_name, timeout, &server_fd, &client_fd) < 0 ) {
    fprintf( stderr, "Error: timeout in opening quick-look socket.\n" );
    return -1;
}
```

The two socket file descriptors **server_fd** and **client_fd** which are returned as function arguments are static variables with a module-wide scope. Because **client_fd** also needs to be accessed by routines outside **ql_server.c**, e.g., routines in **ql_client.c,** it is returned by the function **QL_serverOpen()**.

By default, if an empty socket is read, the reading process will continue until data becomes available in the socket. On the other hand, the **QL** side of the socket channel is polled by the **QL** event loop using a CALL_EXTERNAL call for message flow. Apparently, the polling process will hang up if the reading is blocked due to an empty socket. To prevent a reading process on the socket client from being blocked, a UNIX file control system call is implemented as follows:

```
if ( fcntl( client_fd, F_SETFL, O_NDELAY ) == -1 )
    fprintf( stderr, "Error: unable to unblock socket.\n" );
```

This way, a socket read will return immediately if no message is available. Note that the **fcntl()** system call requires the header file **<fcntl.h>** and the **fcntl()** call is not necessary for the **GUI** side of the socket (server) because the **GUI** event loop is event-driven, not polling.

2. A socket I/O is invoked by the function **QL_serverIO()**. If the read/write flag **rw** is **0**, a socket read is performed using **recv()**. Otherwise, a socket write is executed by calling the low-level socket routine **socket_write()**. The string ***msg** carries the socket message. The code fragment is listed below:

```
if ( rw == 0 ) {
    if ( recv( client_fd, msg, sizeof(msg), 0 ) == -1 ) {
        fprintf( stderr, "Error: failed to receive message\n" );
        return -1;
    }
}
else
    socket_write( client_fd, msg );
```

The socket server on the **GUI** sends a single character to the **QL** socket client to signal a new frame ready for display. This character is "**S**" if the frame is from the spectrometer, and "**C**" if it's from the slit-view camera. In addition, the character "**Q**" is used to notify the **QL** of program exit.

3. The last routine **QL_serverClose()** in the source module disconnects a socket connection by closing the file descriptors **client_fd** and **server_fd**.

3.2 ql_client.c

Like its server counterpart, the socket client source module **ql_client.c** also has three routines which are listed and discussed below:

```
int   QLCom_open( int argc, char *argv[] )     - open a socket client
char *QLCom_io( int agrc, char *argv[] )       - perform a socket I/O
void  QLCom_close( int argc, char *argv[] )    - close socket client
```

1. In fact, there are two socket connections to the **QL**, one from the **GUI** and the other from the image rotator (**IMROT**) program. The second link is used to pass information from the **QL** to the **IMROT**. The socket names are defined by two macros in the beginning of the source file:

```
#define NIRSPEC_SOCKET_NAME    "/tmp/nirspec_ql_socket0"
#define IMROT_SOCKET_NAME      "/tmp/imrot_ql_socket0"
```

In addition, a **STRING** structure is declared for passing an IDL string in the CALL_EXTERNAL call:

```
typedef struct {
    unsigned short slen;
    short stype;
    char *s;
} STRING;
```

IDL calls a routine in a shareable object using the C calling convention **(argc,argv)**. Any routines called by CALL_EXTERNAL should be defined with the following prototype :

```
return_type function( int argc, void *argv[] )
```

where **argc** is the count of optional parameters in the CALL_EXTERNAL call, **argv** is an array of the parameters, and **return_type** is one of the data types which CALL_EXTERNAL may return. The parameter array is passed by reference. This is done by placing a pointer to a **STRING** structure as defined above in **argv[i]**.

QLCom_open() first casts the pointer in **argv[]** to local variables and places the IDL string values into the local C string array **string_array[]**as follows:

```
str_descr = ( STRING * ) argv[0];
n_elements = (short) (*( long * ) argv[1] );

for ( i = 0; i < n_elements; i++, str_descr++ )
    string_array[i] = strdup( str_descr->s );
```

This is also performed by the other two client routines **QLCom_open()** and **QLCom_close()**. If the first parameter **string_array[0]** is "**0**", a socket to the **GUI** is open with the socket file descriptor **fd[0]**; otherwise, a socket to the **IMROT** is open with **fd[1]**, as seen from the following statements:

```
if ( strcasecmp( string_array[0], "0" ) == 0 ) {
    if ( ( status = socket_clientOpen( NIRSPEC_SOCKET_NAME, &fd[0] ) ) < 0 )
        return status;
}
else {
    if ( ( status = socket_clientOpen( IMROT_SOCKET_NAME, &fd[1] ) ) < 0 )
        return status;
}
```

As discusses before, the socket client is polled by the **QL** event loop for messages from the server. Therefore, the following lines must be implemented to prevent the blocking of a socket reading process:

```
if ( fcntl( fd[0], F_SETFL, O_NDELAY ) == -1 ) {
```

```
    fprintf( stderr, "Error: unable to unblock socket.\n" );
    return -1;
}
if ( fcntl( fd[1], F_SETFL, O_NDELAY ) == -1 ) {
    fprintf( stderr, "Error: unable to unblock socket.\n" );
    return -1;
}
```

The routine **QLCom_open()** returns **0** if the socket open is a success.

2. **QLCOm_io()** executes a socket I/O and is straightforward in coding. If the first parameter passed by the CALL_EXTERNAL call is "**0**", a socket read will be performed on the **GUI-QL** link and a character string will be returned by the function (a null string is return if the socket is empty). If it's "**1**", the second parameter **string_array[1]** as passed from the IDL structure **STRING** will be sent to the **IMROT**. This is shown from the following lines:

```
if ( strcasecmp( string_array[0], "0" ) == 0 )
    socket_read( fd[0], str );
else if ( strcasecmp( string_array[0], "1" ) == 0 )
    socket_write( fd[1], string_array[1] );
```

3. Finally, the routine **QLCom_close()** closes a socket connection. Similar to the other two routines, a "**0**" in the passed parameter value indicates the socket link to the **GUI**. Otherwise, it's the socket to the **IMROT**.

4 Program Building

The **NIRSPEC** client software in the directory **/kroot/kui/xnirspec** is built with the make file **makefile** located in the same directory. Since the socket server routines in **ql_server.c** are called by the **GUI** program, they are compiled along with other **GUI** source modules as shown below:

```
INCLUDE = /usr/local/dv/include
CC      = cc
CFLAGS  = -g -I$(INCLUDE) -I$(KROOT)/rel/default/include

NAMES   = xnirspec dataviews create_interest callbacks misc \
          cli_server ql_server dcs_util socket
SOURCE  = $(NAMES:%=%.c)
OBJECT  = $(NAMES:%=%.o)
LIBS    = -L$(KROOT)/rel/default/lib -lktl -lktlker -lkcl -ldl -lXm -lucb

TARGET  = xnirspec cnirspec ql_client.so efs_gateway efs_client.so
all: $(TARGET)

# Build GUI application
```

```
xnirspec: $(OBJECT)
      DVlink -o xnirspec $(OBJECT) $(LIBS)
```

where **DVlink** is a DataViews link script for building DVtools-based X-window applications as the **NIRSPEC GUI** program **xnirspec**.

However, the socket client routines must be compiled as a shareable object library in order for the IDL function CALL_EXTERNAL to call. This is done by the following lines:

```
# Build QL socket client sharable library for IDL
ql_client.so: ql_client.o
      ld -G -o $@ ql_client.o socket.o
```

The built shareable object **ql_client.so** resides in the source directory.

5 Program Execution

The socket server routines are invoked with function calls. For instance, the **GUI-QL** socket connection is established in the **GUI** main program as follows:

```
if ( ( ql_fd = QL_serverOpen( 20 ) ) < 0 )
    ERROR( ("Aborted: failed to open socket connection to quick-look.\n"));
```

Note that a 20-second time-out is set for a connection try.

The client routines contained in the shareable object **ql_client.so** are called from IDL programs using the CALL_EXTERNAL function. For instance, the following IDL code makes a call to **QLCom_open()** to open a socket client for the **GUI-QL** link:

```
inp = strarr(2)
inp(0) = '0'
inp(1) = ' '
status = call_external('/kroot/kui/xnirspec/ql_client.so','QLCom_open', $
                       inp, n_elements(inp), /f_value)
```

The string array **inp** contains parameters to be passed to the called function **QLCom_open()**. Only the first element in the array is used here, however. As discussed before, the first parameter determines which socket to open because the **QL** opens two different socket connections when the **NIRSPEC** program starts. The CALL_EXTERNAL call returns the floating point value **status**. A zero value indicates a success as defined in **QLCom_open()**.

Similarly, the following IDL statements call the routine **QLCom_io()** to read the **GUI-QL** socket for an incoming message. For example, if the character string **msg** is '**S**', a new image frame from the spectrometer has arrived in the host computer and is ready for the **QL** to display.

```
inp = strarr(2)
inp(0) = '0'
inp(1) = ' '
msg = call_external('/kroot/kui/xnirspec/ql_client.so','QLCom_io', $
                    inp, n_elements(inp), /s_value)
```