4444444444444444444444444444444444444444444444444444444444444444

NIRSPEC

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

4444444444444444444444444444444444444444444444444444444444444444

Tim Liu **February 10, 1997**

NIRSPEC Software Design Note 7.01
Command Line Interface

1 Introduction

The NIRSPEC software provides a command line driven user interface as well as the graphical user interface (GUI), according to the requirement specifications. The main function of the command line interface (CLI) is to provide the script capability, which is very useful for carrying out repetitive observing sequences like making a mosaic. Common observing routines such as taking dark frames can easily be constructed using the script tool and executed more efficiently. This document outlines the design and prototyping of the NIRSPEC CLI.

2 Design Requirements and Structure

Briefly, the basic design requirements for CLI are:

! script capability with rich programming facilities
! script verification function
! simple command syntax for ease of use
! automatic conversion of NIRSPEC keywords into valid commands
! simple and modular interface to applications

The first three requirements are obvious for a sophisticated and also easy-to-use CLI. The fourth feature allows one to program the keywords directly. The last requirement will make it easy to “plug in” a different command interpreter if needed.

The heart of a CLI is the command interpreter (CI), or parser, which is responsible for syntax check and error message report. Rather than writing our own CI from scratch, we’d like to build the CLI using an existing command language tool. In fact, The Keck Software Coordination Committee (SCC) has recommanded the use of Tcl (Tool Command Language), a popular scripting language good for controlling and extending applications. As a public-domain program developed by John Ousterhout from UC Berkeley, Tcl has a large user community. Its main features include:

! generic programming facilities like variables, conditionals, looping and procedures
! C shell-like command syntax
! easy to implement new commands

! embeddable and simple API

One of the most useful extensions to Tcl is Tk, an X Window toolkit for quick GUI development, though we're not interested in it because we are using DataViews, a more sophisticated GUI builder program.

The basic features provided by Tcl meet our requirements for the CI. Therefore, we've decided to use it in our CLI implementation. Figure 1 is CLI structure diagram based on Tcl. One can see that the Tcl based CLI is built on top of the keyword layer. Under this design structure, all NIRSPEC commands go through the keyword layer since all the keywords are implemented as Tcl commands.

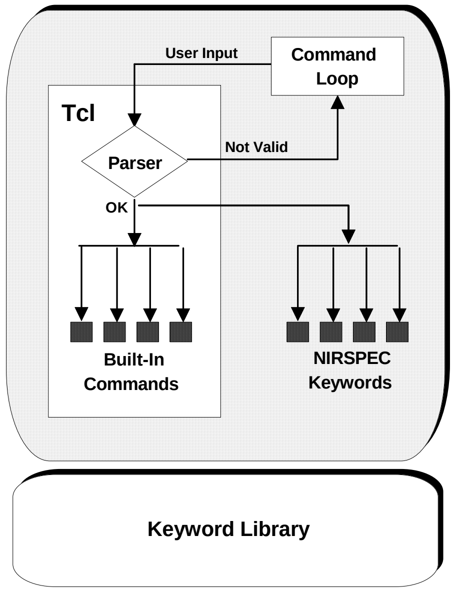

Figure 1 CLI structure

3 Interface

The CLI is implemented as a stand-alone program and runs as a separate process from the GUI. When the NIRSPEC program starts, it opens an xterm as the CLI window where the CLI program is to be run. When the CLI command loop receives a typed-in command string, either a single command or a script name, it passes it to the CI for parsing. If it's an invalid command, an error message will be generated. A Tcl built-in command will be processed by the CI. But for a NIRSPEC keyword command, it will be sent to the GUI via inter-process communication (IPC). When the GUI event loop receives the command string from CLI, it will handle in the same way as a user input from the GUI. The received command will be converted into a keyword/value pair and passed to the NIRSPEC server via RPC. The CLI-GUI interface is illustrated in Figure 2.

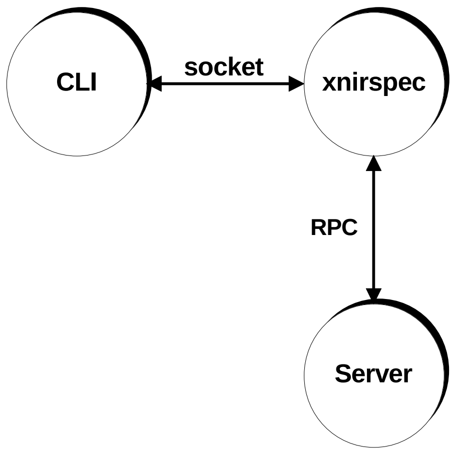

Figure 2 CLI-GUI interface

Unix socket is used for IPC between the CLI and the GUI as seen from Figure 2. To have a better understanding of how the CLI program is implemented, a control flow diagram is shown in Figure 3. The source code for the command loop and parse is rather compact since one just needs to call the Tcl C library routines to perform the task. But each NIRSPEC command (not built-in commands) has to be defined in a new command procedure (it's a C function) to make it valid. This procedure calls a function routine to send out the command string to the GUI for action.

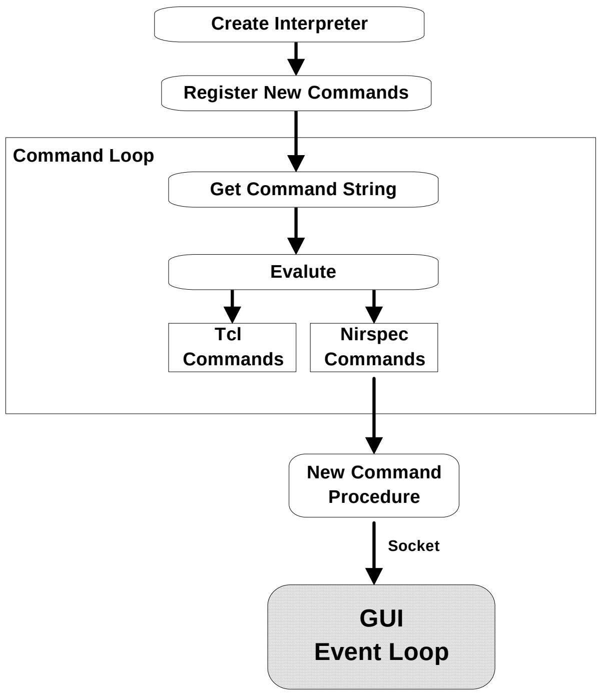

Figure 3 CLI Control Flow Diagram

4 Utilities

A useful CLI should be able to check a script for syntax errors before execution. Unfortunately, Tcl doesn't provide such a function without executing the script. A script verification utility should be developed. To be as simple as possible, the implementation idea is that the "chkscript" command will "source" and execute the script. However, the NIRSPEC commands in the script won't be sent to the NIRSPEC server and therefore no real actions are taken.

Tcl doesn't provide the command recall function either. Since command recall is a very useful feature, we will implement it in the NIRSPEC CLI.

In addition, utilities equivalent to the Keck "show" and "waitfor" will also be implemented.

5 An Example Script

As an example, a dummy Tcl script file is created and listed below. Its function is to make a 3 by 3 mosaic observing sequence. Note that *ra* and *dec* are dummy NIRSPEC telescope offset commands via DCS.

```
#
# NIRSPEC Tcl Script for a 3x3 mosaic
#
# Arguments:
#     os_ra  - offset (arcsecs) in ra direction
#     os_dec - offset (arcsecs) in dec direction
#
# Usage:
#     % source mosaic.tcl
#     % mosaic os_ra os_dec
#
proc mosaic { os_ra os_dec } {
   puts [format "3x3 mosaic with offsets = %d, %d\n" $os_ra $os_dec]

   # loop
   set i 1
   while {$i <= 9} {
      # offset telescope
      if {$i != 1} {
         puts [format "Offsetting to frame #%d ..." $i]
      }

      if {$i == 2} {
```

```
    ra $os_ra
    dec 0
}
elseif {$i == 3} {
    ra 0
    dec $os_dec
}
elseif {$i == 4} {
    ra -{$os_ra}
    dec 0
}
elseif {$i == 5} {
    ra -{$os_ra}
    dec 0
}
elseif {$i == 6} {
    ra 0
    dec -{$os_dec}
}
elseif {$i == 7} {
    ra 0
    dec -{$os_dec}
}
elseif {$i == 8} {
    ra $os_ra
    dec 0
}
elseif {$i == 9} {
    ra $os_ra
    dec 0
}

# start exposure
go 1
# wait until exposure is done
waitfor go 0

if {$i == 9} {
    puts "Back to frame #1 position ..."
    ra -{$os_ra}
    dec $os_dec
}
```

```
        incr i
    }

    puts "Mosaic sequence is finished"
}
```

Using a script is like using a Tcl command, as seen from the comments in the above script. But one has to “source” the script first. This can be done either interactively or from a login file.