Astronomical Data Analysis Software and Systems VII
ASP Conference Series, Vol. 145, 1998
R. Albrecht, R. N. Hook and H. A. Bushouse, eds.

GUI-fying and Documenting your Shell Script

Peter. J. Teuben[1]

Astronomy Department, University of Maryland, College Park, MD 20742, Email: teuben@astro.umd.edu

Abstract.

We describe a simple method to annotate shell scripts and have a preprocessor extract a set of variables, present them to the user in a GUI (using Tcl/Tk) with context sensitive help, and run the script. It then becomes also very easy to rerun the script with different values of the parameters and accumulate output of different runs in a set of user defined areas on the screen, thereby generating a very powerful survey and analysis tool.

1. Introduction

Scripting languages have often been considered the glue between individual applications, and are meant to achieve a higher level of programming.

When individual applications are (tightly) integrated into the scripting language, this offers very powerful scripts, fully graphical user interfaces and a result sometimes indistinguishable from applications. A recent example of this is the glish shell in AIPS++ (Shannon 1996). But of course the drawback of this tight integration is that applications are not always easily accessible to scripts that do not (or cannot) make use of the environment the scripting language was meant for.

Apart from the black art of handcoding, one of the traditional methods to add a GUI to an application is using automatic GUI builders. This has the advantage that application code and user interaction code are more cleanly separated, but this sometimes also limits the flexibility with which the code can be written.

This paper presents a simple implementation where the style of the underlying application is batch oriented, and in fact can be written in any language. The user interface must be cleanly defined in a set of parameters with optional values (e.g., named "*keyword=value*" pairs). Once the input values have been set, the application can be launched, results can be captured and presented in any way the script or application decides.

2. Tcl/Tk: TkRun

The GUI that is created will provide a simple interface to a program that is spawned by the GUI. This program must have a well defined Command Line

Interface (CLI), in the current implementation a "*keyword=value*" interface. Equally well, a Unix-style "*-option value*" could have been used (cf. Appleton's `parseargs` package). The GUI builder, a small 600 line C program called `tkrun`, scans the script for special tags (easily added as comments, which automatically make the script self-documenting), and creates a Tcl/Tk script from which the shell script itself (or any application of choice with a specified CLI) can be launched (see Figure 2 for a schematic).

The added power one gets with this interface builder is the simplified re-execution of the script, which gives the user a powerful tool to quickly examine a complex parameter space of a particular problem.

The input script must define a set of parameters, each with a keyword, an optional initial value, a widget style, usage requirements and one line help. The keyword widget style can be selected from a small set of input styles that standard Tcl/Tk provides (such as generic text entry, file browsers, radio buttons, sliders etc.)

The current implementation has been tested under Tcl/Tk 7.6 as well as 8.0, but is expected to move along as Tcl/Tk is developed further. For example a more modern widget layout technique (`grid` instead of `pack`) should be used. Also keywords cannot have dependencies on each other, for example it would be nice to "grey out" certain options under certain circumstances, or allow ranges of some keywords to depend on the settings of others.

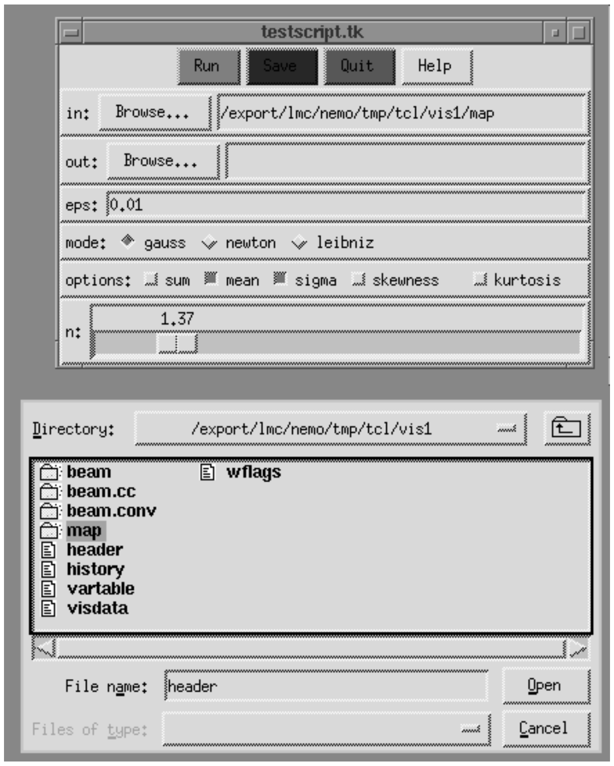
3. Sample Script: testscript

Here is an example header from a C-shell script with which Figure 1 was made. Note that the script must supply a proper "keyword=value" parsing interface, as was done with a simple foreach construct here. The latest version of `tkrun` is available through the NEMO[1] package.

```
#! /bin/csh -f
#                           :: define basic GUI elements for tkrun to extract
#>  IFILE   in=
#>  OFILE   out=
#>  ENTRY   eps=0.01
#>  RADIO   mode=gauss              gauss,newton,leibniz
#>  CHECK   options=mean,sigma      sum,mean,sigma,skewness,kurtosis
#>  SCALE   n=1                     0:10:0.01
#                           :: some one liners
#>  HELP    in      Input filename
#>  HELP    out     Output filename (should not exist yet)
#>  HELP    eps     Initial (small) step
#>  HELP    mode    Integration Method
#>  HELP    options Statistics of residuals to show
#>  HELP    n       Order of polynomial

#                           :: parse named arguments
foreach a ($*)
   set $a
end
```

[1]http://www.astro.umd.edu/nemo/



in: /export/lmc/nemo/tmp/tcl/vis1/map
out:
eps: 0.01
mode: [✓ gauss] newton leibniz
options: sum [✓ mean] [✓ sigma] skewness kurtosis
n: 1.37

Directory: /export/lmc/nemo/tmp/tcl/vis1

beam
beam.cc
beam.conv
map
header
history
vartable
visdata
wflags

File name: header

Figure 1. With the command “`tkrun testscript`” the upper panel is created, providing a simple interface to the “*key=val*” command line interface of the script `testscript` (see below). The lower panel is a standard Tcl/Tk filebrowser that can be connected to keywords that are meant to be files. See Figure 2 for a schematic diagram explaining the interaction between the different programs and scripts.

```
#                             :: actual start of code
echo TESTSCRIPT in=$in out=$out eps=$eps mode=$mode options=$options n=$n
#                             :: legacy script can be inserted here or keyword
#                             :: values can be passed on to another program
```

Acknowledgments. I would like to thank Frank Valdes and Mark Pound for discussing some ideas surrounding this paper, and Jerry Hudson for his Graphics Command Manager FLO.

References

Appleton, Brad (parseargs, based on Eric Allman’s version)

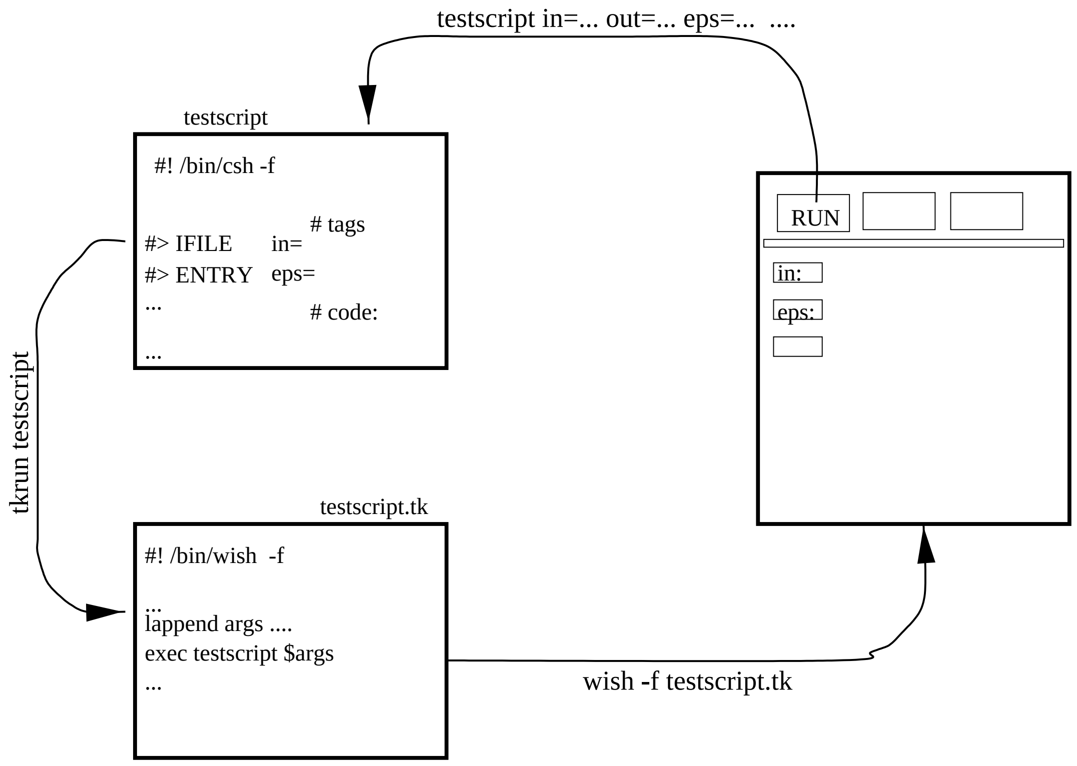


Figure 2. Flow diagram: The command `tkrun` scans the C-shell script `testscript` (top left) for keywords and the Tcl/Tk script `testscript.tk` (bottom left) is automatically written and run. It presents the keywords to the user in a GUI (on the right, see Figure 1 for a detailed view)), of which the “Run” button will execute the C-shell code in the script `testscript`.

Judson, Jerry (FLO: a Graphical Command Manager)

Ousterhout, John, 1994, *Tcl and the Tk Toolkit*, Addison-Wesley

Shannon, P., 1996, in ASP Conf. Ser., Vol. 101, Astronomical Data Analysis Software and Systems V, ed. George H. Jacoby & Jeannette Barnes (San Francisco: ASP), 319

Teuben, P.J., 1995, in ASP Conf. Ser., Vol. 77, Astronomical Data Analysis Software and Systems IV, ed. R. A. Shaw, H. E. Payne & J. J. E. Hayes (San Francisco: ASP), 398