Tcl/Tk for XSPECT[a]

Michael Flynn

Tcl:

Tcl (i.e. Tool Command Language) is an open source scripting language similar to other modern script languages such as Perl or Python. It is substantially more powerful than UNIX/LINUX/POSIX shell script languages such as the Bourne Shell (sh), the C Shell (csh), or the Korn Shell (https://en.wikipedia.org/wiki/Shell_script). Tcl and its associated graphical user interface toolkit, Tk, were developed by John Ousterhout of the University of California. Ousterhout's group subsequently continued development of Tcl/Tk while at Sun Microsystems and later at Scriptics. Continued development is now done by an open source project team. The language has been popular for developing graphic applications and is available as public domain software for almost all computer systems (Linux, Windows and MacOS).

Installation:

Tcl/Tk software can be found at:

http://www.activestate.com/activeTcl

Binary Windows installers are available from this site for both 32 and 64 bit systems as a Free Community Edition. The lab modules were last validated with Version 8.6.4.1 but more recent versions should not be problematic.

Unless Tcl has already been installed on the system being used, download and install the software from ActiveState. The installer should be run with administrative privileges. On Windows7, you should right click and 'run as administrator'. Installing with elevated privilege will allow the registry changes that map Tcl extensions and will configure the un-installer database, making later removal of ActiveTcl easier.

The ActiveState default installation directory is C:/Tcl. During the installation, the installer will ask if you want this changed. I prefer installation in C:/Program Files/Tcl which requires administrator privilege. Note that Tcl uses a forward slash for path delimiters rather than the backward slash used in Windows operating systems.

Console interpreter:

Unlike a programming language, Tcl interprets each command as it is issued. As such it can be used interactively from a console window much like Csh in Linux or for DOS commands in Windows.

---

[a] Revision History:

03/01/99 03/02/03 01/02/04 01/10/05 01/11/13
01/09/14 01/08/16 01/05/18

For Windows, executing *tclsh* or *wish* from the START menu will open a console window from which Tcl or Tcl/Tk commands can be issued. *Wish* includes the Tk commands that place widgets in a window frame. In general, it is best to always use the *wish* interpreter.

If a shell command is not a part of the Tcl/Tk command set but is a part of the operating system command set, the command will be executed. Thus, the command *dir* for Windows will list the contents of the current directory. Frequently, when working interactively in tclsh, the windows *cd* command will be used to change directories.

For tclsh, the '\' character is an escape character. Thus, interpreting path strings such as C:\XIRL\mikef\.. in windows is not possible. Tclsh interprets directory paths using the traditional Unix construction, C:/XIRL/mikef/.. . Alternatively the tcl command;

file join dir1 dir2 dir3

will assemble the indicated directory names into the proper path expression for the operating system being used.

Scripts:

As with other shell languages, Tcl commands can be stored in a file and executed as if the file was a program. For the ActiveState windows installation, if the script file has an a .tcl extension it will be interpreted as Tcl/Tk commands when the file icon is executed. As an example, open a text editor (i.e. notepad, wordpad, emacs, notepad++, ..) and enter two lines with:

console show
puts "Hello myself"

Using 'save as', save the file as temp.tcl. Now execute this file (double click). You should see a small gray window that appears because the script interpreter includes Tk graphic commands (ie. the Wish interpreter). You should also see a command window with 'Hello myself' on the first line. This was output by the *puts* command (i.e put string). In the command window, new commands can be entered at the '%' character command prompt. In the command window, type

%puts "hello you"

and the interpreter will respond. Exit the application by either closing the graphic Tk window or typing *exit* at the command prompt.

To permit execution on a Unix/Linux systems with Tcl/Tk, the mode of the file must be changed using the command 'chmod 755 scriptfile'. The first line of the file should also have the path to the interpreter;

#!/usr/um/bin/wish

Simply typing the filename of the script will then execute the commands in the script.

Tcl commands:

Commands in Tcl will in general always have the following form;

command arg1 arg2 arg3 ...

where command is either one of the standard Tcl commands or may be the name of a procedure defined in the script. White space separates the command name and its arguments and a newline or a semicolon is used to terminate the command.

Tcl uses the pound character, #, for comments. The # specifically must appear at the beginning of a command;

```
#this is a comment
puts "hello myself" ;  # a semicolon ends the command.
```

Interactive Tcl:

For Unix/Linux systems, Tcl can be run interactively simply by typing Tclsh at the normal prompt of a command window. The system will respond with a modified prompt, usually the percent character, %, and the interpreter will be running in the same directory that the user was in when the command was given.

For Windows systems, execute tclsh (or wish) from the START menu. A window will appear in which Tcl (or Tcl/Tk) commands can be issued. The command window will start in the directory that the tclsh or wish program is installed (C:\Program Files\tcl\...).

Any of the Tcl commands can be issued for direct interpretation. In the following we will use a % when suggesting that commands be tested interactively. The interactive mode of Tcl is often useful to test script commands before entering them into an executable script.

Variables within scripts:

Variables can be defined within a script and their value substituted as an argument in a command. To define the value of an argument;

```
%set something  1.234
```

The expression *$something* will then be replaced with 1.234 when used in a command line. For example,

```
%set something  1.234
%puts $something
```

will print 1.234 on the next line of the command window.

Grouping:

The concept of grouping is fundamental in the Tcl language. Either {...} or "..." is used to group words together into one argument that will be a part of a Tcl command. The two forms of grouping behave differently with respect to the substitution of variables (i.e. replacing $var with the value of var:

{...} => no variable substitution within the group
"..." => variables are substituted within the group

For example;

%set something 1.234
%puts "The value is $something"
%puts {The value is $something}

will be seen to produce different results. {...} expressions are often used as a part of control loops such as the *if* command where the bracketed argument may contain many lines of commands. "..." expressions are often used to prepare text strings for output.

Another type of bracketed expression is of the form [...]. In this case, the bracketed words are interpreted as a command and the expression [...] is replaced with the results of the command. This example further illustrates the nesting of executed commands.

%set time [clock format [clock seconds] -format %T]
%puts "The time is $time"

The Tcl clock command has several forms, [clock seconds] returns the number of seconds since a reference time. [clock format ..] formats the value in referenced seconds to a designated format.

Executing xspect programs within a script:

The various programs in the xspect package for modeling x-ray systems are intended for use within scripts. In general, the arguments to a particular program can be passed to the program as a variable. For example, the program to generate an x-ray spectrum requires five values to be entered;

set input "74 12 1 118 1.0 \n"

Since the program reads the five values from a line after the return key, a return is placed in the text string by using the \n at the end. The program can then be executed with the Tcl exec command;

set message [exec spect_gen << $input]

Normally, the programs only return text if there is an error. Tcl provides a command to catch the text returned as a part of standard error reporting;

if [catch {exec spect_gen << $input} result] {
puts stderr "ERROR in SPECT_GEN:\n $result"}

In this example the logical value returned by the catch command is tested by the if command and if true an error message is printed with the text from the *spect_gen* program reported in *$result*. This construction is used later to build script procedures that execute the xspect routines.

Numeric expressions:

To evaluate mathematical expressions, Tcl supports a command called *expr*. For this command, all of the text following the command name are collected and passed to a special argument parser which interprets the mathematic instruction:

%expr 8/4+5

In this example, the text string 8/4+5 would normally be considered to be one command argument. However, for the expr command evaluates the implied math expressions. The math interpreter

recognizes all common arithmetic operators as well as many standard function (sin(x), exp(x), pow(x,y) ...). For example,

```
%set result [expr 1 + sin( exp(2) )]
%puts "1 plus the sin of e^2 is $result"
```

An effective math interpreter distinguishes Tcl from many other shell script languages. The math functions recognized by *expr* are also implemented as a part of math function commands.

Loops within scripts:

Several commands are available in Tcl to produce looping in Tcl scripts. The *while*, *foreach*, and *for* constructions are considered below.

The *while* command:

The general form of the while command is;

```
while booleanExpr body
```

For example;

```
set test 2.0
while {$test <= 100.0} {
        set test [expr $test*2]
        puts $test
}
```

Notice that the left bracket at the end of the first line and the right bracket in the fourth line form the "body" which is one argument value which contains multiple lines of code. The boolean expressions supported are documented with the expr command.

The *foreach* command:

Another type of loop can be used to evaluate the body with a list of particular values. The general form of the command is

```
foreach loopVar valueList body
```

For example;

```
foreach value {0.1 0.3 1.0 3.0 10.0 30.0 100.0} {
        set result [expr log10($value)]
        puts "$value $result"
}
```

The body of the command appears similarly in brackets, *{...}* and has multiple lines of code to be executed. When the commands in the body are read, substitution of argument values for *$arg* is not done. However, when the body is executed as a part of the loop, the *$arg* substitutions are performed. For each execution, *$value* in the body group is sequentially substituted using the list of numbers in *valueList*.

The *for* command:

The *for* command has the general form;

```
for initial test final body
```

For example;

```
for {set i 1} {$i <= 15} {incr i 2} {
        puts $i
}
```

Conditional execution of commands:

The Tcl script language includes conditional commands including *if* and *switch*. We note here only the general construction of the *if* command;

```
if boolean body1 elseif body2 else body3
```

For example;

```
if {$flag == 1} {
        …
} elseif {$flag == 2} {
        …
} else {
        …
}
```

Note that the syntax of the brackets on the lines with the *elseif* or the *else* must be maintained for proper interpretation of the body of commands within each section. Specifically, the newlines are all contained within the brackets of the bodies. Many *elseif* sections can be included (or none).

Opening a file for output:

In Tcl, a file is opened for output using a command that returns a file reference number;

```
set fileID [open temp.dat w]
```

In this expression, *w* indicates that the file will be created or truncated if it already exists. Alternatively, *w+* could be used to append new text to the file. The value returned by this command is assigned to the variable with the name *fileID*.

The file opened by the above command will be created in the active directory that the interpreter is running in. Recall that for Windows, the interactive command window opens in the installation directory. Thus the directory needs to be changed (DOS cd command) or the full path to the file needs to be specified.

Once the file is opened, lines may be written using the puts command.

```
puts $fileID "$value $result"
```

In this case, a line is written to the open file that was assigned the reference number in the variable *fileID*. The line will have the values of the two variables *value* and *result*. When all data has been written, the file should be closed.

```
close $fileID
```

Programming with Tk graphic commands:

Tk extends the Tcl language to include commands which can create a window with widgets on the screen. It is invoked by using the 'wish' interpreter that is the default for windows with the ActiveState installation. For Windows, the 'wish' interpreter can also be initiated as an interactive window using the Start menu. All of the Tcl commands are included in the wish interpreter. To illustrate, start the wish interpreter in the start menu. A command prompt similar to tclsh should appear in a newly created small window. Then try the commands;

%button .hello -text Hello -command {puts stdout "Hi user"}
%button .quit -text Quit -command {destroy .}
%pack .hello .quit -padx 20 -pady 20

The tk command *button* creates button widgets that are referred to as *.hello* and *.quit* within subsequent tk commands. The tk command *pack* assembles widgets into a window. We will defer more detailed discussion of Tk in this document.

For use of xSpect, Tk becomes a convenient way to enter variable values into a script to compute results for different parameter values.

Write an example script:

As an exercise, write a script for to generate a data file. First open a file for output and then use one of the loop constructions to generate a file with two columns of data having about a dozen rows. Assume that the first column is an independent variable and make the value of the second column a dependent value equal to the square of the value in the first column. The data written to this file can then be plotted using a program such as *gnuplot*.

Further information:

On Windows systems with the ActiveState installation, a link to a Tcl/Tk help application will be found in the Start menu. When opened, first expand the "Active Tcl 8.5.13 Documentation Index". Most information of interest will then be in the "TclManual" and "Tk Manual" near the middle of the menu items.

On Unix/Linus systems, most Tcl installations will support man pages describing the execution of the interpreter (man tclsh) as well as man pages for each of the commands.

An excellent book by Brent B. Welch is now published in the fourth edition and covers both introductory as well as advanced aspects of the language up to version 8.4;

"Practical Programming in Tcl and Tk"
ISBN 0-13-038560-3
Fourth Edition, June, 2003
www.beedub.com/book