The Tclsh Spot

By now, most folks have heard about Tcl and Tk, but it seems only fair to introduce a new column with an introduction to the topic.

The Tcl package was developed by Dr. John Ousterhout in the late 80's while he was at UC Berkely. He and his group were developing circuit simulators and found that each project needed a macro language to tune the system. After designing several on-the-fly macro languages, Dr. Ousterhout designed a package that could be merged into the other projects to provide a uniform language across the projects.

Since then, desktop computers have grown from "fast" 33 Mhz 386 processors to "slow" 300 Mhz Pentiums, and Tcl has grown from a simple embeddable macro language into a multi-purpose package.

Describing the modern Tcl is a lot like describing the proverbial elephant.

If you look at Tcl from one angle, it's a scripting language, like `perl`, `awk` or `sh`. If you look at the other side of Tcl/Tk, it's a GUI programming language similar to `Visual Basic` or a multi-platform language, like `Java`. From another angle, Tcl is an interpreter that you can extend with your own commands (or existing libraries). Finally, Tcl is a language toolkit that you can merge into your program.

Just to add a bit to the indescribable nature of Tcl, it's commercially supported freeware. The interpreters (including source code) are supported and made available for free from Scriptics (www.scriptics.com).

The Tcl package comes with two interpreters: `tclsh`, a text-based interpreter suitable for text oriented programs like CGI, shell scripts, or even client-server programs; and `wish`, the same base interpreter as `tclsh` extended with GUI graphics-oriented commands. There are also two libraries, one for Tcl and one for Tk, which let you build your own interpreter, or merge Tcl/Tk into your application.

The ability to merge new object code libraries into the interpreter is the feature that distinguishes Tcl from scripting languages like `sh` and `awk`. This feature lets you merge a vendor supplied library (*.dll or *.a) right into the Tcl or Tk interpreter to create an interpreter with new task-specific commands.

Einar Stefferud sometimes explains that the hallmark of a good internet protocol is that it is simple at the core, with complexities at the edges. The most popular protocols (SMTP, HTTP, NNTP) follow a simple query/response format, with the complexity living in the message content, not the protocol.

Tcl follows a similar pattern: The core Tcl scripting language has a simple and regular syntax with a fairly small number of commands. The complex edges, in this case, are the extensions. The interpreter extensions have new commands that interact a new object code library, while the language core stays the same.

Using Tcl makes it easy to move from problem domain to problem domain. You don't need to learn a whole new language. You just need to learn the new commands for that application.

Several years ago I proved that a novice (me) armed with John Ousterhout's book could create an interpreter with a new set of commands in one evening. Now, with the discussions of extension building in books from Brent Welch, J. A. Zimmer and me, the learning curve may be shorter.

The Tcl syntax is trivial:

- The first word on a line is a command name.
- Words are separated by whitespace.
- Words can be grouped with curly braces ({}) or quotes ("").
- Command lines can be terminated with a semicolon or a newline character.
- A line can be continued across several lines by escaping the newline with a backslash.
- Any words in a command line after the first word are arguments to the command.
- A variable name preceeded by a dollarsign ($) is replaced by the value of the variable.
- A command within square brackets is replaced by the results of evaluating that command. This is similar to how back-quotes (`) are handled by shell scripts.
- A comment starts with a "#" symbol.

So, lets take a cursory look at some Tcl commands. With just 7 commands, we can build a GUI-based calculator.

Assigning Values to Variables

Probably the most used command in Tcl is `set`. The `set` command assigns a value to a variable.

Syntax: `set` *`variableName value`*

```
set foo "bar";          # Assigns the string "bar" to variable foo.
set pi 3.1415;          # Assigns the value 3.1415 to the variable pi.
set x a b;              # An illegal operation,  only one value can be set.
```

Tcl also allows you to append a new string to the value already in a variable. This is done with the `append` command.

Syntax: `append` *`variableName value`*

```
append foo "baz";       # Appends the string "baz" to variable foo.
append pi 19;           # Add two digits of accuracy to the previous value of pi
```

Performing Math

The command to perform arithmetic operations is `expr`, which behaves like the Bourne shell `expr` command. Tcl supports all the math calls in the standard C math library including the trig and exponential functions.

Syntax: `expr` *`algebraicExpression`*

```
set twoPi [expr $pi * 2];       # set the variable twoPi to 2 * pi.
set circumference [expr $twoPi * $radius]; # circumfrence is 2*pi*radius
set area [expr pow($radius, 2) * $pi] ;    # area is pi * radius ^ 2
```

One extremely common math operation is simply incrementing or decrimenting a variable by an integer. To make life a bit simpler, Tcl has a special command for adding a value to a variable: `incr`.

Syntax: `incr` *`variableName value`*

```
set x 2                 # Set x to 2
incr x 4                # Add 4 to the value of x.  X is now 6
incr x -2               # Subtract 2 from the value of x. X is now 4.
```

Looping

Tcl supports a loop-on-counter construct (`for`), a loop-on-test construct (`while`), and a loop-on-list-contents construct (`foreach`).

The calculator example only uses the `foreach`, so that's all I'll describe here.

Syntax: `foreach` *`variableName list { body }`*

The `foreach` command will iterate through the values in the *`list`*. It will evaluate the body after setting the value of the loop variable to the appropriate list element for this iteration.

```
# Initialize the total to 0


set total 0;


# For each value in the list "2 4 9"
#  add that value to the previous value of total


foreach value { 2 4 9 } {
    incr total $value
}
```

GUI Widgets

Tk supports many graphic widgets for building GUI's including a drawable canvas, an editable text window and image operations. For this example, we'll just need two widgets and a geometry manager.

The widget creation commands all follow a common format:

Syntax: *`widgetType .widgetName ?arguments?`*

The *`widgetType`* is the type of widget to create, `button, label, canvas`, etc.

The *`.widgetName`* is a name for this specific instance of the widget. The naming convention for Tk widgets is that widget names must be unique, and must start with a period/lowercase letter pair.

The *`arguments`* let you set various widget configuration options like the text to display, the foreground and background color, size of margins, etc. These are defined as *`-optionName value`* pairs.

All parameters of a Tk widget can be set when the widget is created and modified once a widget exists. However, unlike programming with the X library, the widgets have a set of good defaults, so you don't need to define all the parameters when you create a widget.

Button

One common GUI widget is the `button` widget. This widget will display a string (or graphic) and perform an action when the button is clicked.

Syntax: `button` *`.buttonName ?arguments?`*

A couple commonly used arguments are:

`-text` *`string`* The text to display on the button.

`-command` *body* The body of a command to evaluate when the button is activated.

Label

A label simply displays a string. One of the neat features of the Tk label is that you can link the label to a variable, and it will automatically display the contents of that variable. Your code doesn't need to do anything to update the display.

Syntax: `label` *.labelName ?arguments?*

`-textvariable` *variableName* This label will display the contents of the named variable.
`-text` *string* This label will display a particular string.

Grid

Tk supports three layout managers that let you define how your application should look. For the calculator example, the `grid` manager, which lays out widgets in a spreadsheet style is the simplest to use. The `grid` command defines where a widget will appear and maps the widget onto the display.

Syntax: `grid` *.widgetName ?arguments?*

`-row` *rowNumber* The row for this widget.
`-col` *columnNumber* The column for this widget.
`-columnspan` *number* The number of columns this widget will use.

A Calculator

With those 7 commands, we can construct a little GUI calculator. Now, this is not the last word in online calculators, but it's an example of how little code you need to create a useful Tcl/Tk application. Complete with comments this is 50 lines of code.


```
# Initialize a string that will contain the math operations to perform

set math ""

# Initialize a position counter for the widgets being created.

set pos 0

# Loop through the numbers and operations creating buttons for each widget


foreach val {1 2 3 4 5 6 7 8 9 + 0 - * / } {

    # Create a button
```

```
    # The button names are .b0, .b1, .b2, etc.
    # The text to display is the current item in the list
    # The action for the button is to append that value onto the math string.

    button .b$pos -text $val -command "append math $val"

    # The buttons are displayed in a 3 column wide grid.
    # Calculate the row and column

    set row [expr ($pos) / 3]
    set col [expr ($pos) % 3]

    # And map the widget to the screen

    grid .b$pos -row $row -col $col

    # Increment the position/name counter

    incr pos
}

# The equals button has a different command.
# When the equal button is clicked, it will evaluate the math expression,
# and assign the output to the result variable.
# It then clears the math expression for the next set of calculations.

button .b_eq -text "=" -command {set result [expr $math]; set math ""}
grid .b_eq -row 4 -col 2

# Create two labels to display the math expression and result.

label .math -textvariable math
grid .math -row 5 -column 0 -columnspan 3

label .result -textvariable result
grid .result -row 6 -column 0 -columnspan 3
```

Learning More

If you don't already know Tcl/Tk, you are (I hope) intererested in learning a bit more by now. Here are a few books and websites that will get you started.

Tcl and the Tk Toolkit John Ousterhout	The definitive book, but somewhat dated.
Practical Programming in Tcl and Tk Brent Welch	An excellent book for the experienced programmer.
Graphical Applications with Tcl & Tk Eric Foster-Johnson	A good introductory book
Tcl/Tk for Real Programmers Clif Flynt	I think it's a good book, but I may be biased.

Here are some sites with general Tcl/Tk information:

http://www.scriptics.com	The Scriptics homepage. Up to date information on the state of Tcl, free source code and supported binary downloads, for sale development utilities, training, and support and pointers to Tcl/Tk resources.
http://www.tclconsortium.org	The Tcl/Tk Consortium homepage. The Tcl/Tk Consortium is a non-profit organization of Tcl advocates with a charter to make Tcl known and available to the computing community. The website includes links to resources, information, and a chance to buy pre-compiled versions of Tcl and Tcl extensions for popular platforms.
http://Starbase.NeoSoft.COM/~claird/comp.lang.tcl/	One of the best collections of pointers to Tcl "stuff", ranging from discussions of Tcl fine points to tutorials, books, articles and FAQ's.

And, finally, some sites with online or CAI Tcl/Tk instruction:

http://www.msen.com/~clif/TclTutor.html	The TclTutor interactive computer-based training package for Win 95/NT, Unix, and Macintosh.
http://hegel.ittc.ukans.edu/topics/tcltk/tutorial-noplugin/index.html	Robert Hill, Shyamalan Pather, and Matt Peters created this 13 lesson tutorial on the Tcl language.
http://www.dci.clrc.ac.uk/Publications/Cookbook/index.html	This is an excellent and complete tutorial by Lakshmi and Venkat Sastry. It covers Tcl, Tk, and building extensions.
http://www.cujo.com/tcl_tut.html	William Ho (bill@technologyarchitects.com) has written a concise intro duction to the Tcl language,