Chapter 2

Common Tcl Extensions

Although originally intended as a simple tool command language, Tcl was designed to be easily extensible by means of procedures that can bind to compiled C routines.

Soon Tcl became kind of a cult, as programmers around the globe implemented and made available extension libraries for different purposes. Although these libraries are still called extensions because they are not part of the Tcl language itself, many extension packages have become so closely associated with Tcl that they are considered almost a standard Tcl feature.

This chapter provides a brief introduction to the commonly available extension packages that are included with SGITCL.

Tcl—Tool Command Language

Tcl is an interpreted programming language much like the Bourne shell or the C shell. Unlike these shells, Tcl uses curly braces instead of single quotes to guard against variable, command, and backslash substitution. Also, Tcl uses square brackets instead of backquotes to perform command substitution. Tcl's expression and control flow syntax resembles the C shell more closely than the Bourne shell.

Table 2-1 Delimiters: Tcl Versus Shell

Tcl	Shell	What it Does
{ }	' '	prevents variable, command, or backslash substitution
[ ]	` `	performs command substitution

Strings are the only data type in Tcl, although numeric calculation is possible with the **expr** function. The Tcl language contains a collection of list manipulation facilities including append, insert, search, replace, join, split, and sort procedures.

New procedures can be written in Tcl using the **proc** keyword, or bound to compiled C or C++ procedures for greater efficiency. In fact Tcl is implemented as a library of C procedures, as are most Tcl extensions.

You can run extended Tcl interactively using either the *tclsh* or *sgitcl* command; see the tclsh(1) or sgitcl(1) reference pages for details. There is no way to run unextended Tcl by itself using the SGITCL facility, and really no reason to do so.

Tcl Syntax and Semantics

These eleven rules govern Tcl's syntax and semantics:

1. A Tcl script is a string containing one or more commands. Semicolons and newlines are command separators unless quoted as described below. Close brackets are command terminators during command substitution (see below) unless quoted.
2. A command is evaluated in two steps. First, the Tcl interpreter breaks the command into words and performs substitutions as described below. These substitutions are performed in the same way for all commands. The first word locates a procedure to execute the command, then all words of the command are passed to the procedure, which is free to interpret each of its words in any way it likes (variable name, list, integer, or Tcl script). Different commands interpret their arguments differently.
3. Words of a command are separated by combinations of blanks and tabs. Newlines are command separators.
4. If the first character of a word is a double quote ("), then the word is terminated by the next double quote character. If semicolons, close brackets, or white space characters (including newlines) appear between the quotes, then they are treated as ordinary characters and included in the word. Command substitution, variable substitution, and backslash substitution are performed on the characters between the quotes. The double quotes are not retained as part of the word.
5. If the first character of a word is an open brace ({), then the word is terminated by the matching close brace (}). Braces nest within the word—for each open brace there must be a close brace. (However, if an open brace or close brace within the word is quoted with a backslash, then it is not counted in locating the matching close brace). No substitutions are performed on characters between braces except for backslash-newline substitutions described in Table 2-2, nor do newlines, semicolons, close brackets, or white space receive any special interpretation. A word consists of the characters between the outer braces, not including the braces themselves.

6. If a word contains an open bracket ([), then Tcl performs command substitution. To do this it invokes the Tcl interpreter recursively to process the characters following the open bracket as a Tcl script. The script may contain any number of commands and must be terminated by a close bracket (]). The result of the script is the result of its last command, which is substituted into the word in place of the brackets and all of the characters between them. There may be any number of command substitutions in a single word. Command substitution is not performed on words enclosed in braces.

7. If a word contains a dollar sign ($), then Tcl performs variable substitution. The dollar sign and following characters are replaced in the word by the value of a variable. There may be any number of variable substitutions in a word. Variable substitution is not performed on words enclosed in braces. Variable substitution can take any of the following forms:

$*name*	The name of a scalar variable; *name* is terminated by any character that is not a letter, digit, or underscore.
$*name*(*index*)	Gives the *name* of an array variable and the *index* to an element within that array; *name* must contain only letters, digits, and underscores. Command substitutions, variable substitutions, and backslash substitutions are performed on the characters of *index*.
${*name*}	The name of a scalar variable, which may contain any characters whatsoever except for close braces.

8. If a backslash (\) appears within a word, then backslash substitution occurs. In all cases but those described below, the backslash is dropped and the character after is treated as an ordinary character and included in the word. This allows characters such as double quotes, close brackets, and dollar signs to be included in words without triggering special processing. Backslash substitution is not performed on words enclosed in braces except for backslash-newline as described in Table 2-2. The following table lists backslash sequences that are handled specially, along with the value that replaces each sequence:

Table 2-2 Backslash Sequences in Tcl

Sequence	Meaning
\a	audible alert (bell) (0x7)
\b	backspace (0x8)
\f	form feed (0xc)
\n	newline (0xa)

Table 2-2 (continued) Backslash Sequences in Tcl

Sequence	Meaning
\r	carriage-return (0xd)
\t	tab (0x9)
\v	vertical tab (0xb)
<newline>	A single space character replaces the backslash, newline, and all white space after the newline. This backslash sequence is unique in that it is replaced in a separate pre-pass before the command is actually parsed. This means that it will be replaced even when it occurs between braces, and the resulting space will be treated as a word separator if it isn't in braces or quotes.
\\	backslash
OOO	the digits *OOO* (one to three of them) give the octal value of the character.
\x*HH*	the hexadecimal digits *HH* specify the value of the character. Any number of digits may be present, and are interpreted up to the first non-hex character; leading digits are discarded if they overflow the data type.

9. If a sharp (#) appears at a point where Tcl is expecting the first character of the first word of a command, then the sharp and the characters that follow it, up through the next newline, are treated as a comment and ignored. The comment character has significance only when it appears at the beginning of a command.
10. Each character is processed exactly once by the Tcl interpreter as part of creating the words of a command. For example, if variable substitution occurs, then no further substitutions are performed on the value of the variable; the value is inserted into the word verbatim. If command substitution occurs, then the nested command is processed entirely by the recursive call to the Tcl interpreter. No substitutions are performed before making the recursive call and no additional substitutions are performed on the result of the nested script.
11. Substitutions do not affect the word boundaries of a command. For example, during variable substitution the entire value of the variable becomes part of a single word, even if the variable's value contains spaces.

This example shows Tcl syntax in action:

```
tclsh> set w {chardonnay riesling sauvignon}
tclsh> set r {cabernet pinot zinfandel}
tclsh> concat $r $w
cabernet pinot zinfandel chardonnay riesling sauvignon
tclsh> set wines [concat $r $w]
```

```
tclsh> lsort $wines
cabernet chardonnay pinot riesling sauvignon zinfandel
```

TclX—General Purpose Extensions

Extended Tcl, or TclX, is a set of extensions to Tcl. TclX is oriented towards system programming tasks and large application development. Extended Tcl provides many interfaces to the operating system that Tcl does not, and is upward compatible with Tcl. You can run extended Tcl interactively using either the *tclsh* or *sgitcl* command. In fact *tclsh* is a link to *sgitcl*. See the tclX(3Tcl) reference page for more information.

TclX offers the following features, which plain Tcl does not:

- a shell (*tclsh*) for developing and executing Tcl programs
- a code library facility for building large applications
- access to POSIX system calls
- file I/O commands
- time and date facilities
- string and character manipulation
- *awk*-like pattern scanning
- extended list manipulation commands
- keyed lists (similar to C structures in functionality)
- command for accessing TCP/IP servers
- XPG internationalization commands
- debugging and profiling facilities
- a help system

You can use TclX in conjunction with Tk and Tm, described below.

Tk—User Interface Toolkit

Tk is a toolkit for the X Window System™, accessible from Tcl scripts and implemented as a library of C procedures on top of *libX11*. Because Tk does not require a Motif license from the Open Software Foundation, it runs on freeware systems such as Linux and FreeBSD, and also runs in a limited fashion on Windows™ and Macintosh® systems.

New applications can be created easily by writing short scripts for the windowing shell *wishx*, which provides access to TclX and TkX, an extended form of Tk. Note that the unextended windowing shell *wish* is not available with SGITCL.

Tk was written by John Ousterhout, and is described in his book, *An Introduction to Tcl and Tk*. The Tk widgets are probably the biggest reason for the popularity of Tcl, because they constitute a high-level GUI programming language.

Tm—Tcl Motif

Tcl Motif, or Tm, is a binding of the Tcl language to the real Motif library. Motif widgets are in wide use commercially, and constitute a diverse, popular, and highly functional graphical user interface. Tcl Motif allows programmers to employ Motif widgets instead of the less sophisticated Tk widgets from high-level scripts. The style of programming in Tcl Motif is similar to the style of Tk, although the two interfaces are not compatible.

New Tm applications can be created easily by writing short scripts for the Motif toolkit windowing shell *moat*, which provides access to TclX and Tcl Motif. The function of *moat* is similar to that of *wishx* except that *moat* is for Tm while *wishx* is for Tk.

SGITCL provides access from Tcl Motif to custom IRIX features, such as help menus in ViewKit style, support for the icon-panel library, SGm widgets, Dt combo boxes, and the XbaeMatrix widget. See Chapter 3 for more details.

Expect—Remote Control

Expect is a Tcl-based program for automating remote and interactive programs. Useful for system administration, the *expect* command allows you to write a script that controls interaction with other programs. Scripts know what to expect from a program and what the correct responses should be. An interpreted language provides branching and

high-level control structures to direct the dialogue. In addition, the user can take control and interact directly when desired, then return control to the script.

In general, the *expect* command is useful for running any application that requires interaction between the program and the user. All it requires is for the programmer to characterize this interaction systematically See the expect(1) reference page for more details about this facility.

Expectk—Remote Control Tk

The *expectk* command is similar to *expect*, but it also includes support for Tk widgets. SGITCL includes *expectk* as a link to *wishx*.

See the expectk(1) reference page for more details about Expect with Tk support.

wishx—Windowing Shell

The *wishx* windowing shell allows you to create Tcl/Tk user interfaces by writing short scripts. See the section “Tk—User Interface Toolkit” on page 8 for details.

See the wishx(1) reference page for more details about this facility.

moat—Motif and Tcl Shell

The Motif toolkit windowing shell *moat* allows you to create Tcl Motif user interfaces by writing short scripts. See the section “Tm—Tcl Motif” on page 8 for details.

See the moat(1) reference page for more details about this facility.

incrTcl—Object-Oriented Tcl

Object-oriented Tcl, or incrTcl, is named after the expression [incr Tcl]. This is Tcl syntax for “increment the Tcl variable by one.” This play on words is similar to the name of the C++ language, which is C syntax for “increment the C variable by one.”

SGITCL includes *itcl*, a library of object-oriented extentions to Tcl. They are available from the following interfaces: *sgitcl* (linked to *tclsh* and *wishx*) and *moat*.

See the incrTcl(3Tcl) reference page for more information about incrTcl.

Oratcl and Sybtcl—Database Access

SGITCL includes the libraries *Oratcl* and *Sybtcl* for access to Oracle and Sybase databases from within Tcl. They are available from the *sgitcl* interface (linked to *moat* and *tclsh*) and from *wishx*.

See the Oratcl(3Tcl) reference page for information about Tcl extensions for ORACLE® database access.

See the Sybtcl(3Tcl) reference page for information about Tcl extensions for Sybase® database access.

GLXaux and glxwin

SGITCL includes *GLXAux*, a library of Tk bindings to interface with the GL graphics library, and **glxwin**, a Tk procedure to create and manipulate the GLXwin graphical window widget. These are available from the following interfaces: *sgitcl* (linked to *tclsh* and *wishx*) and *moat*.

See the glxwin(3Tk) reference manual page for information about the **glxwin** procedure.

See the GLXAux(3Tk) reference manual page for information about the *GLXAux* library.