Tcl 8.6 Reference Guide

Tcl/Tk program designed and created by
John Ousterhout **<ouster*(at)*scriptics*(dot)*com>**

Tcl/Tk 8.0 reference guide contents written by
Paul Raines **<raines*(at)*slac*(dot)*stanford*(dot)*edu>**
Jeff Tranter **<tranter*(at)*pobox*(dot)*com>**

Reference guide format designed and created by
Johan Vromans **<jvromans*(at)*squirrel*(dot)*nl>**

Reference guide reduced to Tcl only and updated to Tcl 8.6 by
Peter Kamphuis **<quickref*(at)*campacasa*(dot)*eu>**

The PDF and LaTeX sources of this reference guide can be found at
https://www.campacasa.eu/tcl-quick-reference.html

Contents

Conventions 2
1. Basic Tcl Language Features 2
2. Tcl Special Variables 3
3. Operators and Expressions 3
4. Control Statements 4
5. Tcl Interpreter Information 5
6. Strings and Binary Data 7
7. Lists 12
8. Arrays 15
9. Dictionaries 16
10. System Interaction 18
11. File Information 20
12. File Input/Output 23
13. Channels 26
14. Compression/Decompression Operations 28
15. Packages 32
16. Namespaces 33
17. Multiple Interpreters 35
18. Coroutines 40
19. HTTP/1.1 Protocol 40
20. Object Oriented Tcl 41
21. Other Tcl Commands 43
22. Pattern Globbing 47
23. Regular Expressions 47
Command Index 51

Tcl Reference Guide Revision 8.6.1 (9/2024)

Conventions

fixed	Literal text.
this	Variable text, i.e. things you must fill in.
word	A keyword, i.e. a word with a special meaning.
[ ]	An optional part (unless command substitution with "**[***command***]**" is meant).
...	A repetition.

Several Tcl (sub) commands support the switch or option **--** (double hyphen). It can be used as last switch or option, to allow the next arguments to start with a hyphen.

See *command*(n) manual pages or **https://www.tcl.tk/man/tcl8.6/** for more details on the various Tcl commands. A lot of information from the Tcl command manual pages was taken as input for this reference guide.

1. Basic Tcl Language Features

A Tcl script is a string containing one or more commands, which each are broken into words and undergoing substitions as described below. See **Tcl**(n) manual page for details.

; or **<newline>**	Command separator.
\	Command or line continuation if last character in line.
"hello $a"	Quoting with substitution.
{hello $a}	Quoting with no substitution (deferred substitution).
{*}*word*	Argument expansion. If a word starts with the string "**{*}**" followed by a non-whitespace character, then the leading "**{*}**" is removed and the rest of the word is parsed and substituted as any other word. After substitution, the word is parsed as a list, and its words are added to the command being substituted.
[expr 1+2]	Command substitution.
$*name*	Substitution with value of scalar variable with given name. Braces can be used to protect the variable name (as in **${***name***}xyz**).
$*name***(***index***)**	Substitution with value of element in array variable.
char	Backslash substitution (see below).
#	Comments out rest of line (if first non−whitespace character). Use "**; #**" for a comment at end of line after a command.

The only data type in Tcl is a string. Tcl is Unicode-aware, strings and string lengths go by character, not by byte. Some commands, however, will interpret arguments as numbers/boolean in which case the formats are:

Integer:	**123 -456 0xff** (hex) **0o17 0377** (both octal) **0b0101** (binary)
Floating Point:	**2.1 .3 4. -6e4 7.91e+16**
Boolean:	**false true 0 1 no yes off on** (also upper case)

Tcl makes the following backslash substitutions:

a	audible alert (U+000007)	**<newline>***whiteSpace*	a single space (line continuation)
b	backspace (U+000008)	\\	a backslash
f	form feed (U+00000C)	*ooo*	octal value (*o*=0-7, range 000-377)
n	newline (U+00000A)	**x***hh*	hexadecimal value (*h*=0-9, a-f)
r	carriage return (U+00000D)	**u***hhhh*	Unicode character (*h*=0-9, a-f)
t	horizontal tab (U+000009)	**U***hhhhhhhh*	Unicode character (*h*=0-9, a-f)
v	vertical tab (U+00000B)		

2. Tcl Special Variables

argc	Number of arguments to **tclsh**.
argv	List of arguments to **tclsh**.
argv0	The script that **tclsh** started executing or the name by which **tclsh** was invoked.
auto_path	List of directories to search during auto-load operations.
env	Array where each element name is an enviroment variable.
errorCode	Error code information from the last Tcl error.
errorInfo	Describes the stack trace of the last Tcl error.
tcl_interactive	Contains 1 if running interactively, 0 otherwise.
tcl_library	Location of standard Tcl libraries.
tcl_nonwordchars	A regular expression, controlling what are "non-word" characters.
tcl_patchLevel	Current patchlevel of Tcl interpreter in the form *x.y.z*.
tcl_pkgPath	List of directories to search for package loading.
tcl_platform	Array with elements **byteOrder**, **debug**, **engine**, **machine**, **os**, **osVersion**, **pathSeparator**, **platform**, **pointerSize**, **threaded**, **user**, and **wordSize**.
tcl_precision	Number of significant digits to retain when converting floating-point numbers to strings (default 0, meaning using as few as possible). Don't change, legacy only.
tcl_rcFileName	User-specific startup file to source upon initialization.
tcl_traceCompile	Level of tracing information output during bytecode compilation.
tcl_traceExec	Level of tracing information output during bytecode execution.
tcl_version	Current version of Tcl interpreter in the form *x.y*.
tcl_wordchars	A regular expression, controlling what are "word" characters.

See **tclvars**(n) manual page for details.

3. Operators and Expressions

The **expr** command recognizes the following operators, in decreasing precedence order:

- + ~ !	Unary minus, unary plus, bitwise NOT, logical NOT.
******	Exponentiation.
*** / %**	Multiply, divide, remainder.
+ -	Add, subtract.
<< >>	Bitwise shift left, bitwise shift right.
< > <= >=	Boolean comparisons.
== !=	Boolean equal, not equal.
eq ne	Boolean string equal, not equal.
in ni	In list, not in list.
&	Bitwise AND.
^	Bitwise exclusive OR.
|	Bitwise (inclusive) OR.
&&	Logical AND.
||	Logical OR.
x ? y : z	If **x != 0**, then result is **y**, else **z**.

All operators support integers. All support floating point except **~**, **%**, **<<**, **>>**, **&**, ^, and **|**. Boolean operators can also be used for string operands, in which case string comparison will be used. This will occur if any of the operands are not valid numbers. The **&&**, **||**, and **?:** operators have "lazy evaluation", as in C.

Possible operands are numeric values, Tcl variables (with **$**), strings in double quotes or braces, Tcl commands in brackets, and the following math functions:

abs *arg*	**ceil** *arg*	**exp** *arg*	**isqrt** *arg*	**pow** *x y*	**sqrt** *arg*
acos *arg*	**cos** *arg*	**floor** *arg*	**log** *arg*	**rand**	**srand** *arg*
asin *arg*	**cosh** *arg*	**fmod** *x y*	**log10** *arg*	**round** *arg*	**tan** *arg*
atan *arg*	**double** *arg*	**hypot** *x y*	**max** *arg ...*	**sin** *arg*	**tanh** *arg*
atan2 *y x*	**entier** *arg*	**int** *arg*	**min** *arg ...*	**sinh** *arg*	**wide** *arg*
bool *arg*					

See **expr**(n), **mathfunc**(n) and **mathop**(n) manual pages for details.

4. Control Statements

Commands that control the flow of a Tcl script by conditional or repeated (looping) execution.

break Abort innermost containing loop command. Returns a 3 (**TCL_BREAK**) result code, causing a break exception to occur.

continue
Skip to the next iteration of innermost containing loop command. Returns a 4 (**TCL_CONTINUE**) result code, causing a continue exception to occur.

exit [*returnCode*]
Terminate the process, returning *returnCode* (an integer which defaults to 0) to the system as the exit status.

for *start test next body*
Looping command where *start*, *next*, and *body* are Tcl command strings, and *test* is an expression string. Example:

```
for {set x 0} {$x<10} {incr x} {
    puts "x is $x"
}
```

foreach *varname list body*
The Tcl command string *body* is evaluated for each item in the string *list* where the variable *varname* is set to the item's value. Example:

```
foreach x {a b c d e f} {
    puts "x: $x"
}
```

foreach *varlist1 list1* [*varlist2 list2 ...*] *body*
Same as above, except during each iteration of the loop, the variables in *varlistN* are set to consecutive values from *listN*. Empty values are assigned to *varlistN* if *listN* has less elements than other lists. Examples:

```
set x {}
foreach {i j} {a b c d e f} {
    lappend x $j $i
}
# 3 iterations, x = "b a d c f e"
set x {}
foreach i {a b c} j {d e f g} {
    lappend x $i $j
}
# 4 iterations, x = "a d b e c f {} g"
```

if *expr1* [**then**] *body1* [**elseif** *expr2* [**then**] *body2 ...*] [[**else**] *bodyN*]
If expression string *expr1* evaluates true, Tcl command string *body1* is evaluated. Otherwise if *expr2* is true, *body2* is evaluated, and so on. If none of the expressions evaluate to true then *bodyN* is evaluated. Examples:

```
if {$var1 == 1} { puts "var1 is one" }
if {$var2 == 1} {
    puts "var2 is one"
} elseif {
          $var2 == 2 ||
          $var2 == 3
} then {
    puts "var2 is two or three"
} else {
    puts "var2 is not one, two or three"
}
```

return [*option value* . . .] [*result*]
Return from a procedure, or set return code of a script. The returned *result* defaults to an empty string. Possible options are **-code** *code*, **-errorcode** *list*, **-errorinfo** *info*, **-errorstack** *list*, **-level** *level*, and **-options** *options*.

switch [*options*] [--] *string* [**{**]*pattern1 body1* [*pattern2 body2* . . .] [**}**]
The *string* argument is matched against each of the *patternN* arguments in order. The *bodyN* of the first match found is evaluated. If no match is found and the last pattern is the keyword **default**, its *bodyN* is evaluated. A body specified as "**-**" will use the body of the next pattern. Possible options are **-exact** (the default), **-glob**, **-regexp**, **-nocase**, **-matchvar** *varName* (only with **-regexp**), and **-indexvar** *varName* (only with **-regexp**). Examples:

```
set foo "abc"
switch abc a - b {expr {1}} $foo {expr {2}} default {expr {3}}
# result: 2
switch -glob aaab {
    a*b     -
    b       {expr {1}}
    a*      {expr {2}}
    default {expr {3}}
}
# result: 1
```

while *test body*
Execute the Tcl command string *body* as long as expression string *test* evaluates to true. Example:

```
set x 0
while {$x<10} {
    puts "x: $x"
    incr x
}
```

5. Tcl Interpreter Information

The **info** command provides information about various internals of the Tcl interpreter. Following subcommands (which may be abbreviated) are available:

info args *procName*
Return a list containing the names of the arguments to procedure *procName*, in order.

info body *procName*
Return the body of procedure *procName*.

info class *subcommand class* [*arg* . . .]
Return information about the class *class*. The *subcommand*s are described in section "Object Oriented Tcl" under "Class Introspection".

info cmdcount
Return the total number of commands that have been invoked in this interpreter.

info commands [*pattern*]
Return list of all Tcl commands visible the current namespace (built-ins and procs), optionally string matching *pattern*.

info complete *command*
Return 1 if *command* is a complete Tcl command, 0 otherwise. Complete means having no unclosed quotes, braces, brackets or array element names

info coroutine
Return the name of the currently executing **coroutine**, empty string otherwise.

info default *procName arg varName*
Return 1 if procedure *procName* has a default for argument *arg* and places the value in variable *varName*. Return 0 if there is no default.

info errorstack *interp*
Return, in a form that is programmatically easy to parse, the function names and arguments at each level from the call stack of the last error in the given *interp*, or in the current one if not specified.

info exists *varName*
Return 1 if the variable *varName* exists in the current context and has been defined by being given a value, 0 otherwise.

info frame [*number*]
Provides access to all frames on the stack. If *number* is specified, then the result is a dictionary containing the location information for the command at the *number*ed level on the stack.

info functions [*pattern*]
Return list of all math functions, optionally string matching *pattern*.

info globals [*pattern*]
Return list of global variables, optionally string matching *pattern*.

info hostname
Return name of computer on which interpreter was invoked.

info level [*number*]
Without *number* return the stack level of the invoking procedure. Or return a list with name and arguments of procedure invoked at stack level *number*.

info library
Return name of library directory where standard Tcl scripts are stored. This is the value of the **tcl_library** variable and may be changed by setting that variable.

info loaded [*interp*]
Return list describing packages loaded into *interp*, defaulting to any interpreter.

info locals [*pattern*]
Return list of local variables, optionally string matching *pattern*.

info nameofexecutable
Return full path name of binary from which the application was invoked.

info object *subcommand* [*arg* . . .]
Return information about the object *object*. The *subcommand*s are described in section “Object Oriented Tcl” under “Object Introspection”.

info patchlevel
Return the value of the global variable **tcl_patchLevel**, which holds the exact version of the Tcl library in *major.minor.patchLevel* form by default.

info procs [*pattern*]
Return list of Tcl procedures in current namespace, optionally string matching *pattern*.

info script [*fileName*]
Return name of Tcl script currently being evaluated. Can be set to *fileName* for the duration of the active invocation.

info sharedlibextension
Return extension used by platform for shared objects (for example, **.so**.

info tclversion
Return the value of the global variable **tcl_version**, which holds the version of the Tcl library in *major.minor* form by default.

info vars [*pattern*]
Return list of currently-visible variables, optionally string matching *pattern*.

6. Strings and Binary Data

Commands that process text strings or binary data. Note that Tcl is Unicode-aware, strings and string lengths go by character, not by byte.

append *varName* [*value* . . .]
Appends all of the given *value* arguments to the string stored in *varName*. If *varName* does not exist, it is given a value equal to the concatenation of all the *value* arguments.

binary decode *format* [**-strict**] *data*
Return the binary version of *data* encoded as *format*. Supported formats are **base64**, **hex** and **uuencode**.

binary encode *format* [*-option value* . . .] *data*
Return a readable string of binary *data* encoded as *format*. Supported formats and options are:

base64	**-maxlen** *length*	No line splitting by default.
	-wrapchar *character*	Newline by default.
hex	No options.	
uuencode	**-maxlen** *length*	61 by default, valid range is 5 to 85.
	-wrapchar *character(s)*	Newline by default, acceptable are zero or more \x09 (TAB), \x0B (VT), \x0C (FF), \x0D (CR) followed by zero or one newline \x0A (LF).

binary format *formatString* [*arg* . . .]
Return a binary string representation of *arg*s composed according to *formatString*, a sequence of zero or more field codes each followed by an optional integer count or *****. The possible field codes are:

a	chars (null padding)	**w**	64-bit int (little-endian)
A	chars (space padding)	**W**	64-bit int (big-endian)
b	binary (low-to-high)	**m**	as **w|W**, but native byte order
B	binary (high-to-low)	**f**	float
h	hex (low-to-high)	**r**	float (little-endian)
H	hex (high-to-low)	**R**	float (big-endian)
c	8-bit int	**d**	double
s	16-bit int (little-endian)	**q**	double (little-endian)
S	16-bit int (big-endian)	**Q**	double (big-endian)
t	as **s|S**, but native byte order	**x**	nulls
i	32-bit int (little-endian)	**X**	backspace
I	32-bit int (big-endian)	**@**	absolute position
n	as **i|I**, but native byte order		

binary scan *string formatString* [*varName* . . .]
Extracts values into *varName*'s from binary *string* according to *formatString*.

Return the number of values extracted. Field codes are the same as for **binary format**, except for:

a	chars (no stripping)	**A**	chars (stripping)	**x**	skip forward

encoding convertfrom [*encoding*] *data*
Return *data* converted to Unicode from the specified *encoding* (system encoding by default) as string.

encoding convertto [*encoding*] *string*
Return *string* converted from Unicode to the specified *encoding* (system encoding by default) as a sequence of bytes.

encoding dirs [*directoryList*]
Set search path for additional ***.enc** encoding data files, or return list of directories in search path if *directoryList* is omitted.

encoding names
Return list of available encodings.

encoding system [*encoding*]
Set the system encoding, or return the current system encoding if *encoding* is omitted.

format *formatString* [*arg* . . .]
Return a formatted string generated in the ANSI C **sprintf**-like manner. Placeholders in *formatString* have the form **%**[*argpos***$**][*flag*][*width*][**.***prec*][**h|l|ll**]*char* where *argpos*, *width*, and *prec* are integers and possible values for *flag* are:

–	Left-justified.	*space*	Space padding.	**#**	Alternate output.
+	Always signed.	**0**	Zero padding.		

and possible values for *char* are:

d	Signed decimal.	**X**	Unsigned HEX.	**e**	Float (0e0).
u	Unsigned decimal.	**b**	Unsigned binary.	**E**	Float (0E0).
i	Signed decimal.	**c**	Int to char.	**g**	Auto float (f or e).
o	Unsigned octal.	**s**	String.	**G**	Auto float (f or E).
x	Unsigned hex.	**f**	Float (fixed).	**%**	Plain **%**.

regexp [*switches*] [--] *exp string* [*matchVar*] [*subMatchVar* . . .]
Return 1 if the regular expression *exp* matches part or all of *string*, 0 otherwise. If specified, *matchVar* will be set to all the characters in the match and the following *subMatchVar*'s will be set to matched parenthesized subexpressions. The following switches are supported:

–about	Don't match RE, but instead return a list containing information about the RE for debugging purposes.
–all	Causes the regular expression to be matched as many times as possible in the string, returning the total number of matches found. Any match variables will contain information for the last match only.
–expanded	Enables use of the expanded regular expression syntax where whitespace and comments are ignored.
–indices	Instead of storing the matching characters from *string* in *matchVar* and *subMatchVar*s, each variable will contain a list of two decimal strings giving the indices in *string* of the first and last characters in the matching range of characters.
–inline	The command will return a list with data that otherwise would be placed in match variables. Match variables may not be

specified.

-line Enables newline-sensitive matching. "**[^**" bracket expressions and "**.**" will not match newline, "^" matches an empty string after any newline in addition to its normal function, and "**$**" matches an empty string before any newline in addition to its normal function.

-linestop Changes the behavior of "**[^**" bracket expressions and "**.**" so that they stop at newlines.

-lineanchor Changes the behavior of "^" and "**$**" (the "anchors") so they match the beginning and end of a line respectively.

-nocase Causes upper-case characters in *string* to be treated as lower case during the matching process.

-start *index* A character index offset into the string to start matching the regular expression at.

regsub [*switches*] [--] *exp string subSpec* [*varName*]

Replaces the first portion of *string* that matches the regular expression *exp* with *subSpec*. If *varName* is specified, it will contain the result and the number of replacements made is returned. If *varName* is not specified, the result is returned. Back references can be made in *subSpec*. The following switches are supported:

-all Substitute all ranges in *string* that match *exp*.

-expanded Enables use of the expanded regular expression syntax where whitespace and comments are ignored.

-line Enables newline-sensitive matching. "**[^**" bracket expressions and "**.**" will not match newline, "^" matches an empty string after any newline in addition to its normal function, and "**$**" matches an empty string before any newline in addition to its normal function.

-linestop Changes the behavior of "**[^**" bracket expressions and "**.**" so that they stop at newlines.

-lineanchor Changes the behavior of "^" and "**$**" (the "anchors") so they match the beginning and end of a line respectively.

-nocase Upper-case characters in *string* will be converted to lower-case before matching against *exp*; however, substitutions specified by *subSpec* use the original unconverted form of *string*.

-start *index* A character index offset into the string to start matching the regular expression at.

scan *string formatString* [*varName* . . .]

Extracts values into given variables using ANSI C **sscanf** behavior. Return the number of values extracted, or -1 if nothing could be extracted. If no *varName* is specified, return a list with the extracted data, or an empty string if nothing could be extracted.

Placeholders have the form **%**[*argpos***$**][*****][*width*][**h|L|l|ll**]*char* where ***** is for discard, *argpos*, and *width* are integers and possible values for *char* are:

d	decimal integer	**u**	decimal (unsigned)	**e**,**f**,**g**,**E**,**G**	float
o	octal integer	**i**	any integer	**[***chars***]**	chars in given range
x,**X**	hex integer	**c**	char to int	**[^***chars***]**	chars not in range
b	binary integer	**s**	string (non-blank)	**n**	no input scanned

The **string** command will perform several string operations. Following subcommands (which may be abbreviated) are available:

string bytelength *string*
Return the number of bytes used to represent *string* in memory. **Note:** This subcommand is **deprecated** and likely will be removed from a future Tcl release. Refer to the **string**(n) manual page for more information.

string cat [*string* . . .]
Concatenates the given *string*s and return the resulting compound string. Without any *string*, return an empty string.

string compare [**-nocase**] [**-length** *int*] *string1 string2*
Return -1, 0, or 1, depending on whether *string1* is lexicographically less than, equal to, or greater than *string2*. Optionally comparing in case-insensitive manner or only comparing the first *int* characters.

string equal [**-nocase**] [**-length** *int*] *string1 string2*
Return 1 if *string1* and *string2* are identical, or 0 when not. Optionally comparing in case-insensitive manner or only comparing the first *int* characters.

string first *needleString haystackString* [*startIndex*]
Return index in *haystackString* of first occurance of *needleString*, or -1 if not found. Optionally at or after index *startIndex*.

string index *string charIndex*
Return the *charIndex*'th character in *string*, or an empty string if *charIndex* doesn't fit to *string*.

string is *class* [**-strict**] [**-failindex** *varName*] *string*
Return 1 if *string* is a valid member of the specified character *class*, or 0 when not. If **`-strict`** is specified, then an empty string returns 0, otherwise an empty string will return 1 on any class. If **`-failindex`** is specified, upon failure variable *varName* will contain the index in *string* where *class* is no longer valid. Following character classes are recognized (class name can be abbreviated):

`alnum`	Any Unicode alphabet or digit character.
`alpha`	Any Unicode alphabet character.
`ascii`	Any character with a value less than \u0080 (7-bit ASCII range).
`boolean`	Any boolean form.
`control`	Any Unicode control character.
`digit`	Any Unicode digit character (includes characters outside of the [0-9] range).
`double`	Any floating point form.
`entier`	Any arbitrily sized integer value.
`false`	Any boolean form that is false.
`graph`	Any Unicode printing character, except space.
`integer`	Any 32-bit integer value. Returns 0 and *varName* is -1 upon overflow.
`list`	Any proper list structure.
`lower`	Any Unicode lower case alphabet character.
`print`	Any Unicode printing character, including space.
`punct`	Any Unicode punctuation character.
`space`	Any Unicode whitespace character.
`true`	Any boolean form that is true.
`upper`	Any Unicode upper case alphabet character.
`wideinteger`	Any wide integer. Returns 0 and *varName* is -1 upon overflow.

wordchar	Any Unicode word character (any alphanumeric character and any connector punctuation characters, like underscore).
xdigit	Any hexadecimal digit character ([0-9A-Fa-f]).

string last *needleString haystackString* [*lastIndex*]
Return index in *haystackString* of last occurance of *needleString*, or -1 if not found. Optionally at or before index *lastIndex*.

string length *string*
Return the number of **characters** in *string*.

string map [**-nocase**] *mapping string*
Replaces substrings in *string* based on the list of key-value pairs in *mapping* and return the result. Iteration over *string* is only done once, any key appearing first will be replaced first. Optionally comparing in case-insensitive manner.

string match [**-nocase**] *pattern string*
Return 1 if glob *pattern* matches *string*, 0 otherwise. Optionally comparing in case-insensitive manner.

string range *string first last*
Return characters from *string* at indices *first* through *last* inclusive.

string repeat *string count*
Return *string* repeated *count* number of times.

string replace *string first last* [*newString*]
Remove characters from *string* at indices *first* through *last* inclusive, and return the result. If *newString* is specified, it replaces the removed characters.

string reverse *string*
Return *string* with its characters in reverse order.

string tolower *string* [*first*] [*last*]
Return new string formed by converting all characters in *string* to lower case. Optionally only converting from index *first* to index *last*.

string totitle *string* [*first*] [*last*]
Return new strings formed by converting the first character in *string* to title case (upper case), and all further characters to lower case. Optionally only converting from index *first* to index *last*.

string toupper *string* [*first*] [*last*]
Return new string formed by converting all characters in *string* to upper case. Optionally only converting from index *first* to index *last*.

string trim *string* [*chars*]
Return new string formed by removing from *string* any leading or trailing characters present in the set *chars* (defaults to white space).

string trimleft *string* [*chars*]
Return new string formed by removing from *string* any leading characters present in the set *chars* (defaults to white space).

string trimright *string* [*chars*]
Return new string formed by removing from *string* any trailing characters present in the set *chars* (defaults to white space).

string wordend *string charIndex*
Return index of character just after last one in word at *charIndex* in *string*. **Note:** This subcommand is **deprecated** and likely will be removed from a future Tcl release. Refer to the **string**(n) manual page for more information.

string wordstart *string charIndex*
Return index of first character of word at *charIndex* in *string*. **Note:** This

subcommand is **deprecated** and likely will be removed from a future Tcl release. Refer to the **string**(n) manual page for more information.

subst [**-nobackslashes**] [**-nocommands**] [**-novariables**] *string*
Return result of backslash, command, and variable substitutions on *string*. Each may be turned off by switch.

Following string processing commands are provided through the **Tcl** library:

tcl_endOfWord *string charIndex*
Return the index of the first end-of-word location after *charIndex* in *string* (-1 if not found).

tcl_startOfNextWord *string charIndex*
Return the index of the first start-of-word location after *charIndex* in *string* (-1 if not found).

tcl_startOfPreviousWord *string charIndex*
Return the index of the first start-of-word location before *charIndex* in *string* (-1 if not found).

tcl_wordBreakAfter *string charIndex*
Return the index of the first word boundary after *charIndex* in *string* (-1 if not found).

tcl_wordBreakBefore *string charIndex*
Return the index of the first word boundary before *charIndex* in *string* (-1 if not found).

Following string processing commands are provided through the **tcl** namespace:

::tcl::prefix all *table string*
Return a list of all elements in *table* (a list of strings) that begin with the prefix *string*.

::tcl::prefix longest *table string*
Return the longest common prefix of all elements in *table* (a list of strings) that begin with the prefix *string*.

::tcl::prefix match [*options*] *table string*
If *string* equals one element in *table* (a list of strings) or is a prefix to exactly one element, the matched element is returned. If not, the result depends on the **-error** option. Options are:

`-exact` Accept only exact matches.

`-message` *string*
Use *string* in the error message at a mismatch. Default is “option”.

`-error` *options*
The *options* are used when no match is found. If *options* is empty, no error is generated and an empty string is returned. Otherwise the *options* are used as **return** options when generating the error message.

Note: String indices start at 0 and the word **`end`** may be used to reference the last character in the string. Computations in the form **`end-`***N* are possible.

7. Lists

Commands that process lists. Since the only data type in Tcl is a string, lists can be represented by strings with space separated content. For example, “**`[list arg1 arg2 arg3]`**” is equal to “**`"arg1 arg2 arg3"`**”.

concat [*arg arg* . . .]
Return the concatenation of each list *arg* as a single list.

join *list* [*joinString*]
Return the string created by joining all elements of *list* with *joinString* (a space character by default).

lappend *varName* [*value* ...]
Appends each *value* as element to the end of the list stored in *varName*. List *varName* is created with the *value* elements if it doesn't exist yet.

lassign *list varName* [*varName* ...]
Assign *list* elements to variables *varName*. Too many *varName* will be empty, too few *varName* will return unassigned elements. Can be used as "shift" command as known from shell languages, like in:

```
set ::argv [lassign $::argv argumentToReadOff]
```

lindex *list* [*index* ...]
Return the value of element at *index* in *list*. Without *index* return *list*. Multiple *index* allow to select from sublists.

linsert *list index element* [*element* ...]
Return a new list formed by inserting given new elements before element at *index* in *list*.

list [*arg* ...]
Return a new list formed by using each *arg* as an element.

llength *list*
Return number of elements in *list*.

lmap *varName list body*
Execute *body* with each element of *list* assigned to loop variable *varName*. Returns an accumulator list, which gets the results of *body* appended if *body* completes normally. The **break** and **continue** statements can be used in *body* to have it not complete normally.

lmap *varList1 list1* [*varList2 list2* ...] *body*
Same as above, except during each iteration of the loop, the variables in *varlistN* are set to consecutive values from *listN*. Empty values are used in *varlistN* if *listN* doesn't have enough elements.

lrange *list first last*
Return a new list from slice of *list* at indices *first* through *last* inclusive.

lrepeat *number element1* [*element2* ...]
Return a new list consisting of *number* times the sequence of *elementN*.

lreplace *list first last* [*element* ...]
Return a new list formed by replacing zero or more elements with indices *first* through *last* in *list* with zero or more other *element*s.

lreverse *list*
Return a new list consisting of elements of *list* in reverse order.

lsearch [*options*] *list pattern*
Return the index of first element in *list* that matches *pattern* (or -1 for no match). With options **-all** or **-inline** a list or matching value is returned (or empty string for no match). The following options are supported:

-exact	*Pattern* is a literal string to be matched exactly.
-glob	*Pattern* is a glob-style pattern to be matched. This is the default.
-regexp	*Pattern* is treated as regular expression.
-sorted	Expects *list* to be sorted and will use a more efficient search algorithm. Behaves like **-exact**.
-all	Return a list of all matching indices.

-inline	Return the matching value instead of its index.
-not	Negates the sense of the match.
-start *index*	Search *list* starting at position *index*.
-ascii	Examine *list* elements as Unicode strings (option name is for backward compatibility).
-dictionary	Compare *list* elements using dictionary-style comparisons (see **lsort**).
-integer	Compare *list* elements as integers.
-nocase	Compare in case-insensitive manner. No effect with **-dictionary**, **-integer** or **-real**.
-real	Compare *list* elements as floating-point values.
-decreasing	When used with **-sorted**, assume *list* elements sorted in decreasing order.
-increasing	When used with **-sorted**, assume *list* elements sorted in increasing order. This is the default.
-bisect	Implies **-sorted**. Return the last index where the element is less than or equal to the pattern (or greater than or equal for an decreasing *list*).
-index *indexList*	When searching lists of lists, the path of indices within each *list* element defining the term to match against.
-subindices	Returns complete path(s) as index result. Has no effect without **-index**.

lset *varName* [*index* . . .] *newValue*

Replace an element at *index* in the list stored in *varName* with *newValue* (and also return that list). Appends to the list if *index* is equal to the number of elements in **$***varName*. Multiple *index* allow to assign to sublists. Without *index* replaces the old value of *varName*.

lsort [*options*] *list*

Return a new list formed by sorting *list* according to *options*. These are:

-ascii	Use string comparison with Unicode code-point collation order (option name is for backward compatibility). This is the default.
-dictionary	Like **-ascii** but ignores case and is number smart. Overrides **-nocase**.
-integer	Convert list elements to integers and use integer comparison.
-real	Convert list elements to floating-point values and use floating comparison.
-command *command*	Uses *command* for comparison, which takes two arguments and returns an integer less than, equal to, or greater than zero.
-increasing	Sort *list* in increasing order. This is the default.
-decreasing	Sort *list* in decreasing order.
-indices	Returns a list of indices into *list* in sorted order instead of the values themselves.
-index *indexList*	Assumes elements of *list* to be sublists. Sorts on the *indexList*th element of each sublist.
-stride *strideLength*	Assumes groups of *strideLength* elements and sorts on their

first element. If used with **-index**, sorts on the specified index of each group.

-nocase Compare in case-insensitive manner.

-unique Uniquify the sorted list. The last duplicate is kept.

split *string* [*splitChars*]
Return a list formed by splitting *string* at each character in *splitChars*. Splits on white-space by default.

Note: List indices start at 0 and the word **end** may be used to reference the last element in the list. Computations in the form **end-***N* are possible.

8. Arrays

Associative arrays in Tcl can be indexed by arbitrary strings, and are stored and retrieved without any specific order. Array element values are directly accessed with "**$arrayvar(element)**".

The **array** command will perform several array operations. Following subcommands (which may be abbreviated) are available:

array anymore *arrayName searchId*
Return 1 if anymore elements are left to be processed in array search *searchId* on *arrayName*, 0 otherwise. *SearchId* is the return value from a previous **array startsearch** invocation.

array donesearch *arrayName searchId*
Terminate the array search *searchId* on *arrayName*, and destroy the state associated with that search. Returns an empty string. *SearchId* is the return value from a previous **array startsearch** invocation.

array exists *arrayName*
Return 1 if *arrayName* is an array variable, 0 otherwise.

array get *arrayName* [*pattern*]
Return a list where each odd element is an element name in *arrayName* and the following even element its corresponding value. Optionally returning only for array elements that string match *pattern*.

array names *arrayName* [*mode*] [*pattern*]
Return a list of all element names in *arrayName*, optionally string matching *pattern*. *Mode* may be one of:

-exact *Pattern* is a literal string to be matched exactly.

-glob *Pattern* is a glob-style pattern to be matched. This is the default.

-regexp *Pattern* is treated as regular expression.

array nextelement *arrayName searchId*
Return the name of next element in *arrayName* for the search *searchId*. *SearchId* is the return value from a previous **array startsearch** invocation.

array set *arrayName list*
Set values of elements in *arrayName* for *list* in **array get** format.

array size *arrayName*
Return number of elements in *arrayName*.

array startsearch *arrayName*
Return a search id to use for an element-by-element search of *arrayName*.

array statistics *arrayName*
Return statistics about the distribution of data within the hashtable that represents the array.

array unset *arrayName* [*pattern*]
Unset all elements in the array *arrayName*, optionally string matching *pattern*. Returns an empty string.

Following array command is provided through the **Tcl** library:

parray *arrayName* [*pattern*]
Print to standard output the names and values of all element names in *arrayName*, optionally string matching *pattern*.

9. Dictionaries

Dictionaries are values that contain an efficient, order-preserving mapping from arbitrary keys to arbitrary values. Each key in the dictionary maps to a single value. They have a textual format that is exactly that of any list with an even number of elements, with each mapping in the dictionary being represented as two items in the list.

The **dict** command will perform several dictionary operations. Following subcommands (which may be abbreviated) are available:

dict append *dictVariable key* [*string* ...]
Append the given *string*(s) to the value that the given *key* maps to in the dictionary value contained in the given variable, writing the resulting dictionary value back to that variable. Non-existent keys are treated as if they map to an empty string. Returns the updated dictionary value.

dict create [*key value* ...]
Return a new dictionary that contains each of the key/value mappings listed as arguments.

dict exists *dictValue key* [*key* ...]
Return a boolean value indicating whether the given key exists in the given dictionary value. Multiple *key*s can be specified to address nested dictionaries.

dict filter *dictValue filterType arg* [*arg* ...]
Return a new dictionary that contains just those key/value pairs that match the specified filter type on the given dictionary value. Supported filter types are:

dict filter *dictValue* **key** *globPattern*
Match only those key/value pairs whose keys string match the given *globPattern*.

dict filter *dictValue* **script** **{***keyVar valueVar***}** *script*
Evaluate the given *script* (returning a boolean value) for each key/value pair (assigned to the given *keyVar* and *valueVar* variables). Match only those key/value pairs for which *script* returns true. The *script* can return with a condition of **TCL_BREAK** or **TCL_CONTINUE** accordingly.

dict filter *dictValue* **value** *globPattern*
Match only those key/value pairs whose values string match the given *globPattern*.

dict for **{***keyVar valueVar***}** *dictValue body*
Iterate over the given dictionary value, setting the *keyVar* and *valueVar* variables for each key/value pair, and evaluate the given script *body* (like **foreach**). Returns an empty string. The given script can generate a **TCL_BREAK** or **TCL_CONTINUE** result accordingly.

dict get *dictValue* [*key* . . .]
Get the value of the given *key* in the given dictionary value. Multiple *key*s allow getting values in nested dictionaries. If no *key*s are given, return a list containing key/value pairs (like **array get**).

dict incr *dictVariable key* [*increment*]
Add the given *increment* value (default 1) to the value that the given *key* maps to in the dictionary value contained in the given variable, writing the resulting dictionary value back to that variable. Non-existent *key*s are treated as if they map to 0. Returns the updated dictionary value.

dict info *dictValue*
Return information about the given dictionary value.

dict keys *dictValue* [*globPattern*]
Return a list of all keys in the given dictionary value, optionally only those keys string matching *globPattern*.

dict lappend *dictVariable key* [*value* . . .]
Append the given items to the list value that the given key maps to in the dictionary value contained in the given variable, writing the resulting dictionary value back to that variable. Non-existent keys are treated as if they map to an empty list. Returns the updated dictionary value.

dict map {*keyVar valueVar***}** *dictValue body*
Apply a transformation to each element of a dictionary, returning a new dictionary. Script *body* is evaluated with *keyVar* and *valueVar* set to each element of the dictionary value contained in the given variable. The script can use **break**, terminating the command and returning the dictionary elements processed so far, and **continue**, aborting the current iteration and not modifying the current element.

dict merge [*dictValue* . . .]
Return a dictionary that contains the contents of each of the *dictValue* arguments. In case two or more dictionaries contain a mapping for the same key, the resulting dictionary maps that key to the value of the last specified dictionary with that key.

dict remove *dictValue* [*key* . . .]
Return a new dictionary that is a copy of the given dictionary value, with the mappings for each *key* listed removed.

dict replace *dictValue* [*key value* . . .]
Return a new dictionary that is a copy of the given dictionary value, replacing or adding the given key/value pairs.

dict set *dictVariable key* [*key* . . .] *value*
Update the dictionary value contained in the given variable by mapping the given *key* to the given *value*. Multiple *key*s allow setting values in nested dictionaries.

dict size *dictValue*
Return the number of key/value mappings in the given dictionary value.

dict unset *dictVariable key* [*key* . . .]
Update the dictionary value contained in the given variable to not contain a mapping for the given *key*. Multiple *key*s allow removing mappings in nested dictionaries. Returns the updated dictionary value.

dict update *dictVariable key varName* [*key varName* . . .] *body*
Execute the Tcl script in *body* with the value for each *key* (of the dictionary value in the given variable) mapped to the variable *varName*. Changes made to the *varName* variable(s) are reflected back to the given dictionary. Returns the result of the the evaluation of *body*.

dict values *dictValue* [*globPattern*]
Return a list of all values in the given dictionary value, optionally only those values

string matching *globPattern*.

dict with *dictVariable* [*key* ...] *body*
Execute the Tcl script in *body* with the value for each *key* (of the dictionary value in the given variable) mapped to a variable with the same name. Multiple *key*s allow nested dictionaries. Returns the result of the the evaluation of *body*.

10. System Interaction

Commands that interact with the operating system.

cd [*dirName*]
Change working directory to *dirName*, or to the home directory (**`$env(HOME)`**) if *dirName* is not given. Returns an empty string.

clock *subCommand* [*parameters*]
Obtain and manipulate dates and times. Any *timeVal* parameter indicates a time expressed as an integer number of seconds since 1 January 1970, 00:00 UTC. For clock arithmetic, formatting, and scanning the following options are supported:

`-gmt` *boolean*
Specifies that a time should be processed in UTC, or defaults to the local time zone. This usage is obsolete; correct would be to use e.g. **`-timezone :UTC`**.

`-locale` *localeName*
Specifies that locale-dependent processing is to be done in the locale identified by *localeName*.

`-timezone` *zoneName*
Specifies that processing is to be done according to the rules for the time zone specified by *zoneName*. Default is to use the current or local time zone.

Following **clock** *subCommands* are available:

clock add *timeVal* [*count unit* ...] [*option value*]
Add an offset to a time *timeVal* and return the result. As *unit* one of the words **`seconds`**, **`minutes`**, **`hours`**, **`days`**, **`weeks`**, **`months`**, or **`years`**, or any unique prefix of such word can be used.

clock clicks [*resolution*]
Return a hi-res system-dependent integer time value. The *resolution* can be specified as **`-milliseconds`** or **`-microseconds`** to obtain a count in milliseconds or microseconds, respectively. The use of *resolution*, however, is obsolete and **clock milliseconds** or **clock microseconds** are the preferred ways of obtaining these counts.

clock format *timeVal* [*option value* ...]
Convert *timeVal* to a human-readable format and return the result as string. The option **`-format`** *format* recognizes the following placeholders (names as for the given locale):

`%a`	weekday (abbr)	`%N`	month (1 – 12)
`%A`	weekday (full)	`%Od,%Oe,%OH,%OI,%Ok,%Ol,%Om,`	
`%b`	month (abbr)	`%OM,%OS,%Ou,%Ow,%Oy`	
`%B`	month (full)		locale as without “O”
`%c`	locale date **`&`** time	`%p`	locale AM/PM
`%C`	century	`%P`	locale am/pm
`%d`	day (01 – 31)	`%Q`	reserved for internal use
`%D`	`%m/%d/%Y`	`%r`	locale 12hr time
`%e`	day (1 – 31)	`%R`	`%H:%M`

%Ec	locale date & time	%s	*timeVal*
%EC	locale era	%S	seconds (00 – 59)
%EE	B.C.E or C.E.	%t	TAB
%Ex	locale alternate date	%T	%H:%M:%S
%EX	locale alternate time	%u	weekday (1 – 7 = *Mon* – *Sun*)
%Ey	locale alternate year (00 – 99)	%U	week (00 – 53)
%EY	locale alternate year (full)	%V	ISO8601 week (01 – 53)
%g	ISO8601 year (00 – 99)	%w	weekday (0 – 6 = *Sun* – *Sat*)
%G	ISO8601 year (full)	%W	week (00 – 53)
%h	month (abbr), same as %b	%x	locale date
%H	hour (00 – 23)	%X	locale time
%I	hour (01 – 12)	%y	year (00 – 99)
%j	day (001 – 366)	%Y	year (full)
%J	Julian day number	%z	time zone (±hhmm)
%k	hour (0 – 23)	%Z	locale time zone name
%l	hour (1 – 12)	%%	literal %
%m	month (01 – 12)	%+	"%a %b %e %T %Z %Y"
%M	minute (00 – 59)		

The default format is "%a %b %d %T %Z %Y".

clock microseconds

Return the current time as an integer number of microseconds.

clock milliseconds

Return the current time as an integer number of milliseconds.

clock scan *inputString* [*option value* . . .]

Scan a time that is expressed as a character string *inputString* and return an integer number of seconds. The option **-format** *format* (see **clock format** above) preferably is used to describe the expected format of *inputString*. Not using that option will request a free-form scan and is deprecated: there are too many ambiguities. The option **-base** *timeVal* specifies that any relative time present in *inputString* is relative to *timeVal*.

clock seconds

Return the current time as an integer number of seconds (*timeVal*).

exec [**-ignorestderr**] [**-keepnewline**] [--] *arg* [*arg* . . .] [**&**]

Execute subprocess using each *arg* as word for a shell pipeline and return results written to standard out, optionally retaining the final newline char. With **-ignorestderr** output to standard error will not be treated as error.

If the last argument is "**&**" execute the pipeline in background and return a list of all subprocess process identifiers. Standard output from the last command in the pipeline will go to the application's standard output unless redirected.

The following constructs can be used to control I/O flow:

|	Pipe (stdout).
|&	Pipe (stdout and stderr).
< *fileName*	Stdin from file.
<@ *fileId*	Stdin from open file.
<< *value*	Pass value to stdin.
> *fileName*	Stdout to file.
2**>** *fileName*	Stderr to file.
>& *fileName*	Stdout and stderr to file.
>> *fileName*	Append stdout to file.
2**>>** *fileName*	Append stderr to file.

>>& *fileName*	Append stdout and stderr to file.
>@ *fileId*	Stdout to open file.
2**>@** *fileId*	Stderr to open file.
2**>**@1	Redirect stderr to stdout (must be at end).
>&@ *fileId*	Stdout and stderr to open file.

Note: Behavior and capabilities of **exec** on Windows platform are different.

glob [*switches*] [--] [*pattern* . . .]

Return a list of all files that match any of the given csh- or bash-style glob patterns. Note that no particular order is guaranteed in the list. The following *switches* are supported.

-directory *directory*	Search for files within the given *directory*.
-join	Treat all *pattern* joined with directory separators as single pattern.
-nocomplain	Allow empty result without error.
-path *pathPrefix*	Search for files with given *pathPrefix*.
-tails	Only return **file tail** of each file found in any **-directory** or **-path** specification.
-types *typeList*	Only return files or directories of certain type. The following types will be ORed: **b** (block special file), **c** (character special file), **d** (directory), **f** (plain file), **l** (symbolic link), **p** (named pipe), or **s** (socket). The following (UNIX) types will be ANDed: **r** (readable), **w** (writable), and **x** (executable).

pid [*fileId*]

Return a list of process ids of all the processes in the pipeline *fileId* if given, otherwise return process id of interpreter process.

pwd Return the absolute path name of the current working directory.

11. File Information

The **file** command provides several operations on a file's name or attributes. Following subcommands (which may be abbreviated) are available:

file atime *name* [*time*]

Return the time *name* was last accessed as seconds since January 1, 1970. Set access time of *name* if *time* is specified.

file attributes *name* [*option* [*value* . . .]]

Return or set platform-specific attributes of *name*. Return a list if no *option* specified. Options for UNIX:

-group [*id*]	Return group name. Specified *id* can be name or id.
-owner [*id*]	Return owner name. Specified *id* can be name or id.
-permissions [*mode*]	Return octal code. Specified *mode* can be octal or symbolic, as known for **chmod**(1).

Options for Windows:

-archive [*value*]	Return the value of or set or clear the archive attribute.
-hidden [*value*]	Return the value of or set or clear the hidden attribute.
-longname	Return the path with each element expanded to its long version. Cannot be set.

-readonly [*value*] Return the value of or set or clear the readonly attribute.

-shortname Return the path with each element replaced with its short (8.3) version. Cannot be set.

-system [*value*] Return the value of or set or clear the system attribute.

Options for MacOS:

-creator [*type*] Return or set the Finder creator type.

-hidden [*value*] Return the value of or set or clear the hidden attribute.

-readonly [*value*] Return the value of or set or clear the readonly attribute.

-rsrclength [**0**] Return the length of the resource fork. Can only be set to 0 (stripping off the resource fork).

file channels [*pattern*]
Return a list of all registered open channels, optionally string matching *pattern*.

file copy [**-force**] [--] *source* [*source* . . .] *target*
Makes a copy of *source* under name *target*. If multiple sources are given, *target* must be a directory. If *source* is a directory, its contents are recursively copied into the *target* directory. Copied soft links are retained. Use **-force** to overwrite existing files.

file delete [**-force**] [--] *pathName* [*pathName* . . .]
Removes given files or directories. Use **-force** to remove non-empty directories.

file dirname *name*
Return a name comprised of all path components in *name* excluding the last element.

file executable *name*
Return 1 if file *name* is executable by current user, 0 otherwise.

file exists *name*
Return 1 if file *name* exists and user has search privileges for the directories leading to it, 0 otherwise.

file extension *name*
Return all characters in *name* after and including the last dot in the last element of *name*. Return empty string if the last element of *name* doesn't contain a dot.

file isdirectory *name*
Return 1 if file *name* is a directory, 0 otherwise.

file isfile *name*
Return 1 if file *name* is a regular file, 0 otherwise.

file join *name* [*name* . . .]
Joins file names using the correct path separator for the current platform.

file link [**-symbolic** | **-hard**] *linkName* [*target*]
Create a link *linkName* pointing to *target*. Return the link target if *target* is not specified (or an error if *linkName* is not a link). On UNIX the default is a symbolic link.

file lstat *name varName*
Same as **file stat** except uses the *lstat* kernel call. If *name* is a symbolic link, the information returned in *varName* is for the link instead of the file it refers to.

file mkdir *dir* [*dir* . . .]
Create each directory specified. Any non-existent parent directories will also be created.

file mtime *name* [*time*]
Return the time *name* was last modified as seconds since January 1, 1970. Set modification time of *name* if *time* is specified.

file nativename *name*
Return the platform-specific name of *name*.

file normalize *name*
Return a unique absolute, resolved and normalized path representation of *name*.

file owned *name*
Return 1 if *name* is owned by the current user, 0 otherwise.

file pathtype *name*
Return one of **absolute**, **relative**, or **volumerelative**.

file readable *name*
Return 1 if *name* is readable by current user, 0 otherwise.

file readlink *name*
Return the value of the symbolic link given by *name*.

file rename [**-force**] [--] *source* [*source* ...] *target*
Rename file or directory *source* to *target*. If *target* is an existing directory, each source file or directory is moved there. The **-force** option forces overwriting of existing files.

file rootname *name*
Return all the characters in *name* up to but not including last dot in the last component of *name*. Return *name* if the last component doesn't contain a dot.

file separator [*name*]
Return the character used to separate path segments on the current platform if no *name* is specified, or the separator of the filesystem responsible for *path*.

file size *name*
Return the size of *name* in bytes.

file split *name*
Return a list whose elements are the path components of *name*. The first list element for an absolute path on UNIX will be "/".

file stat *name varName*
Place results of stat kernel call on *name* in variable *varName* as an array with elements **atime**, **ctime**, **dev**, **gid**, **ino**, **mode**, **mtime**, **nlink**, **size**, **type**, and **uid**. Each element is a decimal string, only **type** is a string as returned by **file type**.

file system *name*
Return a list of one or two elements. The first is the name of the filesystem for *name*, the second represents the type if available.

file tail *name*
Return all characters in *name* after last directory separator (or just *name* if it doesn't contain any separators).

file tempfile [*nameVar*] [*template*]
Create a temporary file and return a read-write channel opened on that file. If *nameVar* is specified, the name of the temporary file will be written into it. With *template* parts of the template of the filename to use can be specified.

file type *name*
Return a string giving the type of *name*. Possible values are **file**, **directory**, **characterSpecial**, **blockSpecial**, **fifo**, **link**, or **socket**.

file volumes
Return a list of absolute paths to the volumes mounted on the system. On UNIX, without any virtual filesystems mounted as root volumes, this is just "/". On Windows this is a list of local drives.

file writable *name*
Return 1 if *name* is writable by current user, 0 otherwise.

12. File Input/Output

Commands to handle file reading and writing operations.

close *channelId* [**r**[**ead**]|**w**[**rite**]]
Close or half-close the open file channel *channelId*. A bidirectional channel can be half-closed by specifying the direction to only close.

eof *channelId*
Return 1 if an end-of-file has occurred on *channelId*, 0 otherwise.

fblocked *channelId*
Return 1 if last read from *channelId* exhausted all available input.

fconfigure *channelId* [*option* [*value* [*option value* . . .]]]
Set or get options for I/O channel *channelId*. Options are:

-blocking *boolean*
Whether I/O can block process.

-buffering full|line|none
How to buffer output.

-buffersize *newSize*
Size of buffer in bytes. Minimum value is 1, maximum value is 1 million.

-encoding *name*
Encoding of the channel. Allows to convert to and from Unicode for use in Tcl. Use **binary** as *name* for reading and writing binary files.

-eofchar *char* **| {***inChar outChar***}**
Character to serve as end-of-file marker.

-translation *mode* **| {***inMode outMode***}**
How to translate end-of-line markers. Modes are **auto**, **binary**, **cr**, **crlf**, and **lf**.

For socket channels the following query options are supported:

-connecting
For client sockets, return 1 if an asynchronous connect is still in progress, 0 otherwise.

-error
Get the current error status as message of the socket (or empty string).

-peername
For client and accepted sockets, return a three element list with address, host name and port to which the peer socket is connected or bound.

-sockname
For client sockets, return a three element list with address, host name and port number for the socket.

For serial device channels the following options are supported (see **open**(n) manual page for details):

-mode *baud*,*parity*,*data*,*stop*
Specifying baud rate (integer), parity (as **n** (none), **o** (odd), **e** (even), **m** (mark), or **s** (space)), number of data bits (5 to 8), and number of stop bits (1 or 2).

-handshake none|rtscts|xonxoff|dtrdsr
Type of handshake control. Type **dtrdsr** only available on Windows. This option cannot be queried.

-queue
This option can only be queried. Return a list of two integers: Current number of bytes in input and output queue.

-timeout *msec*
Set timeout in milliseconds for blocking read operations. On UNIX the granularity is 100 milliseconds. This option cannot be queried.

-ttycontrol {*signal boolean signal boolean . . .* **}**
Set up the handshake output lines. This option cannot be queried.

-ttystatus
This option can only be queried. Return the current modem status and handshake input signals as list of signal and value pairs.

-xchar {*xonChar xoffChar***}**
The software handshake characters.

-pollinterval *msec*
This option is only available on Windows. The maximum time in milliseconds between polling for fileevents. Default is 10 msec.

-sysbuffer *inSize* **| {***inSize outSize***}**
This option is only available on Windows. The size in bytes of Windows system buffers for a serial channel. Default is 4096 bytes.

-lasterror
This option is only available on Windows and can only be queried. Get a list of error details in case **read** or **puts** have returned a file I/O error.

fcopy *inchan outchan* [**-size** *size*] [**-command** *callback*]
Copy data from *inchan* to *outchan* until end-of-file or *size* bytes have been transferred. If **-command** is given, copy occurs in background and runs *callback* when finished, appending number of bytes copied and an optional error message as arguments.

fileevent *channelId* **readable | writable** [*script*]
Evaluate *script* when channel *channelId* becomes readable/writable. Return the current script if *script* is not specified. Delete the event handler if *script* is an empty string.

flush *channelId*
Flushes any output that has been buffered for *channelId*.

gets *channelId* [*varName*]
Read next line from channel *channelId*, discarding the end-of-line character(s). If *varName* is not given, return the line read. If *varName* is given, place the line read in it and return the number of characters read.

open *fileName* [*access* [*permissions*]]
Open a file, serial port, or command pipeline and return a channel identifier. A command pipeline is opened if *fileName* starts with **|**, allowing to write to the command's input pipe or read from its output pipe. The *access* argument can be specified as:

r Read only. File must exist. Default if *access* is not specified.

r+ Read and write. File must exist.

w Write only. Truncate if exists.

w+ Read and write. Truncate if exists.

a Write only. Create new empty file if not existing yet. Access position at end.

a+ Read and write. Create new empty file if not existing yet. Access position at end.

All above access values may have the character **b** added to indicate binary reading or writing (as if configured with **fconfigure -translation binary**). Alternatively the *access* argument can be specified as a list of following flags, of which one must be either **RDONLY**, **WRONLY** or **RDWR**:

RDONLY	Read only.
WRONLY	Write only.
RDWR	Read and write.
APPEND	Access position at end.
BINARY	Binary reading or writing.
CREAT	Create new empty file if not existing yet.
EXCL	If **CREAT** is also specified, an error is returned if the file already exists.
NOCTTY	If the file is a terminal device, prevent the file from becoming the controlling terminal of the process.
NONBLOCK	Prevents the process from blocking while opening the file, and possibly in subsequent I/O operations. Preferably **fconfigure** is used to put a file in nonblocking mode.
TRUNC	Truncate if file exists.

If a new file is created, its permission are set to the conjuction of *permissions* (defaulting to **0666**) and the process umask.

puts [**-nonewline**] [*channelId*] *string*
Write string to *channelId* (defaulting to **stdout**), optionally omitting the end-of-line character at the end.

read [**-nonewline**] *channelId* [*numChars*]
Read all data from *channelId* up to the end of the file, optionally discarding the last character of the file if it is an end-of-line character. If *numChars* is specified, read only this amount of characters (or the remaining characters if fewer are left in the file).

seek *channelId offset* [*origin*]
Change the current access position of *channelId* to *offset* bytes (can be negative) from *origin* which may be **start** (default), **current**, or **end**.

socket [*options . . .*] *host port*

socket -server *command* [*options*] *port*
Open a client or server side network socket connection and return a channel identifier. For a client connection *port* and *host* specify a port to connect to. Possible options are:

-myaddr *addr*	Set network address of client (if multiple available).
-myport *port*	Set connection port of client.
-async	Connect the client socket asynchronously.

If the **-server** option is specified then the new socket will be a server that listens on the given *port*. A *command* is invoked with three arguments: the channel, the address, and the port number.

-myaddr *addr*	Set network address of server (if multiple available).

Socket channels can be configured with **fconfigure** or **chan configure**.

tell *channelId*
Return current access position in *channelId* as byte offset.

13. Channels

The **chan** command provides a unified way to read, write and manipulate channels that have been created with the **open** or **socket** commands, or the default named channels **stdin**, **stdout** or **stderr**. Several operations are also available using a mix of "old" commands (see File Input/Output above).

Following subcommands (which may be abbreviated) are available:

chan blocked *channelId*
: Return 1 if the last input operation on channel *channelId* failed because it would have otherwise caused the process to block, 0 otherwise.

chan close *channelId* [**r**[**ead**]|**w**[**rite**]]
: Close and destroy channel *channelId*. A bidirectional channel can be half-closed by specifying the direction to only close.

chan configure *channelId* [*option* [*value* [*option value* . . .]]]
: Set or get options for channel *channelId*. Options are:

  -blocking *boolean*
  : Whether I/O can block process.

  -buffering full|line|none
  : How to buffer output.

  -buffersize *newSize*
  : Size of buffer in bytes. Maximum value is 1 million.

  -encoding *name*
  : Encoding of the channel. Allows to convert to and from Unicode for use in Tcl. Use the **binary** as *name* for reading and writing binary files.

  -eofchar *char* **|** {*inChar outChar*}
  : Character to serve as end-of-file marker.

  -translation *mode* **|** {*inMode outMode*}
  : How to translate end-of-line markers. Modes are **auto**, **binary**, **cr**, **crlf**, and **lf**.

  For socket channels the following query options are supported:

  -connecting
  : For client sockets, return 1 if an asynchronous connect is still in progress, 0 otherwise.

  -error
  : Get the current error status as message of the socket (or empty string).

  -peername
  : For client and accepted sockets, return a three element list with address, host name and port to which the peer socket is connected or bound.

  -sockname
  : For client sockets, return a three element list with address, host name and port number for the socket.

  For serial device channels the following options are supported (see **open**(n) manual page for details):

  -mode *baud*,*parity*,*data*,*stop*
  : Specifying baud rate (integer), parity (as **n** (none), **o** (odd), **e** (even), **m** (mark), or **s** (space)), number of data bits (5 to 8), and number of stop bits (1 or 2).

  -handshake none|rtscts|xonxoff|dtrdsr
  : Type of handshake control. Type **dtrdsr** only available on Windows. This option cannot be queried.

-queue
This option can only be queried. Return a list of two integers: Current number of bytes in input and output queue.

-timeout *msec*
Set timeout in milliseconds for blocking read operations. On UNIX the granularity is 100 milliseconds. This option cannot be queried.

-ttycontrol {*signal boolean signal boolean . . .* **}**
Set up the handshake output lines. This option cannot be queried.

-ttystatus
This option can only be queried. Return the current modem status and handshake input signals as list of signal and value pairs.

-xchar {*xonChar xoffChar***}**
The software handshake characters.

-pollinterval *msec*
This option is only available on Windows. The maximum time in milliseconds between polling for fileevents. Default is 10 msec.

-sysbuffer *inSize* **| {***inSize outSize***}**
This option is only available on Windows. The size in bytes of Windows system buffers for a serial channel. Default is 4096 bytes.

-lasterror
This option is only available on Windows and can only be queried. Get a list of error details in case **read** or **puts** have returned a file I/O error.

chan copy *inputChan outputChan* [**-size** *size*] [**-command** *callback*]
Copy data from channel *inputChan* to channel *outputChan* until end-of-file or *size* bytes have been transferred. If **-command** is given, copy occurs in background and runs *callback* when finished, appending number of bytes copied and an optional error message as arguments.

chan create *mode cmdPrefix*
Create a new script level channel using the command prefix *cmdPrefix* as its handler. The argument *mode* must be a list containing any of the strings **read** or **write** and specifies how the channel is opened. This channel is called a reflected channel. See **refchan**(n) manual page for details.

chan eof *channelId*
Return 1 if an end-of-file has occurred on channel *channelId*, 0 otherwise.

chan event *channelId event* [*script*]
Arrange for the Tcl script *script* to be installed as a file event handler to be called whenever channel *channelId* enters the state described by *event* (which must be either **readable** or **writable**). Specify an empty string as *script* to delete the current handler. Without *script* returns the currently installed script.

chan flush *channelId*
Flushes any output that has been buffered for channel *channelId*.

chan gets *channelId* [*varName*]
Read next line from channel *channelId*, discarding the end-of-line character(s). If *varName* is not given, return the line read. If *varName* is given, place the line read in it and return the number of characters read.

chan names [*pattern*]
Return a list of all channel names, optionally only those string matching *pattern*.

chan pending *mode channelId*
Depending on whether *mode* is **input** or **output**, return the number of bytes of input or output currently buffered internally for channel *channelId*. Return -1 if the channel was not opened for the mode in question.

chan pipe
Create a standalone pipe whose read- and write-side channels are returned as a 2-element list (read side and write side).

chan pop *channelId*
Remove the topmost transformation from the channel *channelId*, if there is any. Close the channel if there are no transformations added to *channelId*.

chan postevent *channelId eventSpec*
This subcommand is used by command handlers specified with **chan create**. It notifies the channel represented by the handle *channelId* that the event(s) listed in the *eventSpec* have occurred. The argument has to be a list containing any of the strings **`read`** and **`write`**. See **refchan**(n) manual page for details.

chan push *channelId cmdPrefix*
Add a new transformation on top of the channel *channelId*. The *cmdPrefix* argument describes a list of one or more words which represent a handler that will be used to implement the transformation. See **transchan**(n) manual page for details.

chan puts [**-nonewline**] [*channelId*] *string*
Write string to channel *channelId* (defaulting to **`stdout`**), optionally omitting end-of-line character at the end.

chan read [**-nonewline**] *channelId* [*numChars*]
Read all data from *channelId* up to the end of the file, optionally discarding last character of the file if it is a end-of-line character. If *numChars* is specified, read only this amount of characters (or the remaining characters if fewer are left in the file).

chan seek *channelId offset* [*origin*]
Change current access position of *channelId* to *offset* bytes (can be negative) from *origin* which may be **`start`** (default), **`current`**, or **`end`**.

chan tell *channelId*
Return current access position in channel *channelId* as byte offset.

chan truncate *channelId* [*length*]
Set the byte length of the underlying data stream for *channelId* to be *length* (or to the current byte offset if *length* is omitted).

14. Compression/Decompression Operations

The **zlib** command provides access to the compression and check-summing facilities of the Zlib library by Jean-Loup Gailly and Mark Adler.

Following subcommands are available:

zlib compress *string* [*level*]
Return the zlib-format compressed binary data of the binary string in *string*. If present, *level* gives the compression level to use (from 0, which is uncompressed, to 9, maximally compressed).

zlib decompress *string* [*bufferSize*]
Return the uncompressed version of the raw compressed binary data in *string*. If present, *bufferSize* is a hint as to what size of buffer is to be used to receive the data.

zlib deflate *string* [*level*]
Return the raw compressed binary data of the binary string in *string*. If present, *level* gives the compression level to use (from 0, which is uncompressed, to 9, maximally compressed).

zlib gunzip *string* [**-headerVar** *varName*]

Return the uncompressed contents of binary string *string*, which must have been in gzip format. If **-headerVar** is given, store a dictionary describing the contents of the gzip header in the variable called *varName*. The keys of the dictionary that may be present are:

comment	The comment field from the header, if present.
crc	A boolean value describing whether a CRC of the header is computed.
filename	The filename field from the header, if present.
os	The operating system type code field from the header. See RFC 1952 for the meaning of these codes.
size	The size of the uncompressed data.
time	The time field from the header if non-zero, expected to be time that the file named by the **filename** field was modified. Suitable for use with **clock format**.
type	The type of the uncompressed data (**binary** or **text**) if known.

zlib gzip *string* [**-level** *level*] [**-header** *dict*]

Return the compressed contents of binary string *string* in gzip format. If **-level** is given, *level* gives the compression level to use (from 0, which is uncompressed, to 9, maximally compressed). If **-header** is given, *dict* is a dictionary containing values used for the gzip header. The following keys may be defined:

comment	Add the given comment to the header of the gzip-format data.
crc	A boolean saying whether to compute a CRC of the header. Note that if the data is to be interchanged with the **gzip** program, a header CRC should **not** be computed.
filename	The name of the file that the data to be compressed came from.
os	The operating system type code, which should be one of the values described in RFC 1952.
time	The time that the file named in the **filename** key was last modified. This will be in the same as is returned by **clock seconds** or **file mtime**.
type	The type of the data being compressed, being **binary** or **text**.

zlib inflate *string* [*bufferSize*]

Return the uncompressed version of the raw compressed binary data in *string*. If present, *bufferSize* is a hint as to what size of buffer is to be used to receive the data.

zlib push *mode channel* [*options* . . .]

Push a compressing or decompressing transformation onto the channel *channel*. The transformation can be removed again with **chan pop**. The *mode* argument determines what type of transformation is pushed; the following are supported:

compress	A compressing transformation that produces zlib-format data on *channel*, which must be writable.
decompress	A decompressing transformation that reads zlib-format data from *channel*, which must be readable.
deflate	A compressing transformation that produces raw compressed data on *channel*, which must be writable.
gunzip	A decompressing transformation that reads gzip-format data from *channel*, which must be readable
gzip	A compressing transformation that produces gzip-format data on *channel*, which must be writable

inflate A decompressing transformation that reads raw compressed data from *channel*, which must be readable.

The following *options* may be set when creating a transformation with the **zlib push** command:

-dictionary *binData*
: Sets the compression dictionary to use when working with compressing or decompressing the data to be *binData*. Not valid for gzip-format data.

-header *dictionary*
: Passes a description of the gzip header to create, in the same format that **zlib gzip** understands.

-level *compressionLevel*
: The compression level to use (from 0, which is uncompressed, to 9, maximally compressed).

-limit *readaheadLimit*
: The maximum number of bytes ahead to read when decompressing. This option has become **irrelevant**.

Both compressing and decompressing channel transformations add extra configuration options that may be accessed through **chan configure**:

-checksum *checksum*
: Read-only. Gets the current checksum for the uncompressed data that the compression engine has seen so far. Valid for both compressing and decompressing transforms, but not for the raw inflate and deflate formats.

-dictionary *binData*
: Read-write. Gets or sets the initial compression dictionary to use when working with compressing or decompressing the data to be *binData*. Not valid for gzip-format data.

-flush *type*
: Write-only. Flushes the current state of the compressor to the underlying channel. Only valid for compressing transformations. The *type* can be **sync** for a normal flush, or **full** for an expensive flush.

-header *dictionary*
: Read-only. Returns the dictionary describing the header read off the data stream for gzip-data decompressing transforms.

-limit *readaheadLimit*
: Read-write. Used by decompressing channels to control the maximum number of bytes ahead to read from the underlying data source. See above for more information.

Streaming

Create a streaming compression or decompression command, and return the name of the command (*stream* in the below):

zlib stream compress [**-dictionary** *bindata*] [**-level** *level*]
: The stream will be a compressing stream that produces zlib-format output, using compression level *level* as integer from 0 to 9 (if specified), and the compression dictionary *bindata* (if specified).

zlib stream decompress [**-dictionary** *bindata*]
: The stream will be a decompressing stream that takes zlib-format input and produces uncompressed output. If *bindata* is supplied, it is a compression dictionary to use if required.

zlib stream deflate [**-dictionary** *bindata*] [**-level** *level*]
: The stream will be a compressing stream that produces raw output, using

compression level *level* as integer from 0 to 9 (if specified), and the compression dictionary *bindata* (if specified).

zlib stream gunzip
The stream will be a decompressing stream that takes gzip-format input and produces uncompressed output.

zlib stream gzip [**-header** *header*] [**-level** *level*]
The stream will be a compressing stream that produces gzip-format output, using compression level *level* as integer from 0 to 9 (if specified), and the header descriptor dictionary *header* (if specified; for keys see **zlib gzip**).

zlib stream inflate [**-dictionary** *bindata*]
The stream will be a decompressing stream that takes raw compressed input and produces uncompressed output. If *bindata* is supplied, it is a compression dictionary to use.

Streaming compression instance commands:

stream **add** [*option* . . .] *data*
A short-cut for "*stream* **put** [*option* . . .] *data*" followed by "*stream* **get**".

stream **checksum**
Return the checksum of the uncompressed data seen so far by this stream.

stream **close**
Delete this stream and frees up all resources associated with it.

stream **eof**
Return a boolean indicating whether the end of the stream (as determined by the compressed data itself) has been reached.

stream **finalize**
A short-cut for "*stream* **put -finalize {}**".

stream **flush**
A short-cut for "*stream* **put -flush {}**".

stream **fullflush**
A short-cut for "*stream* **put -fullflush {}**".

stream **get** [*count*]
Return up to count bytes from *stream*'s internal buffers with the transformation applied. If *count* is omitted, the entire contents of the buffers are returned.

stream **header**
Return the gzip header description dictionary extracted from the stream. Only supported for streams created with **zlib stream gunzip**.

stream **put** [*option* . . .] *data*
Append the contents of the binary string *data* to *stream*'s internal buffers while applying the transformation. The following options are supported, which are used to modify the way in which the transformation is applied:

-dictionary *binData*
Sets the compression dictionary to use when working with compressing or decompressing the data to be *binData*.

-finalize
Mark the stream as finished, ensuring that all bytes have been wholly compressed or decompressed. For gzip streams, this also ensures that the footer is written to the stream.

-flush
Ensure that a decompressor consuming the bytes that the current (compressing) stream is producing will be able to produce all the bytes that have been compressed so far, at some performance penalty.

-fullflush
: Ensure that not only can a decompressor handle all the bytes produced so far (as with **-flush** above) but also that it can restart from this point if it detects that the stream is partially corrupt. This incurs a substantial performance penalty.

The options **-finalize**, **-flush** and **-fullflush** are mutually exclusive.

stream **reset**
: Put any stream, including those that have been finalized or that have reached eof, back into a state where it can process more data. Throws away all internally buffered data.

Check-summing

zlib adler32 *string* [*initValue*]
: Compute a checksum of binary string *string* using the Adler-32 algorithm. If given, *initValue* is used to initialize the checksum engine.

zlib crc32 *string* [*initValue*]
: Compute a checksum of binary string *string* using the CRC-32 algorithm. If given, *initValue* is used to initialize the checksum engine

15. Packages

The **package** command keeps a simple database of the packages available for use by the current interpreter and how to load them into the interpreter. Typically, only the **package require** and **package provide** commands are invoked in normal Tcl scripts; the other commands are used primarily by system scripts that maintain the package database.

Following subcommands are available:

package forget *package*
: Remove all information about *package* from interpreter.

package ifneeded *package version* [*script*]
: Tell interpreter that *package* with *version* is available if needed, and that the package can be added to the interpreter by executing *script*. Return the current script if *script* is not provided, or an empty string.

package names
: Return a list of all packages in the interpreter that are currently provided or have an **package ifneeded** script available.

package prefer [**latest** | **stable**]
: Return or set the **package require** selection logic mode.

package present [**-exact**] *package* [*requirement*]
: Equivalent to **package require**, but does not try and load *package* if not already loaded.

package provide *package* [*version*]
: Tell interpreter that *package version* is now provided. Without *version*, the currently provided version of *package* is returned, or an empty string.

package require *package* [*requirement* . . .]
: Tell interpreter that a suitable *package* must be provided. A suitable package must satisfy at least one of the *requirement*s as per **package vsatisfies** rules. The version number of the package loaded is returned.

package require [**-exact**] *package version*
: Tell interpreter that *package* with the exact *version* must be provided.

package unknown [*command*]
: Specify a “last resort” Tcl command to invoke during **package require** if no

suitable version of a package can be found. The command will get the desired package name and requirements appended. Return the current command if *command* is not provided, or an empty string.

package vcompare *version1 version2*
Return -1 if *version1* is earlier than *version2*, 0 if equal, and 1 if later.

package versions *package*
Return a list of all version numbers of *package* with a **package ifneeded** script.

package vsatisfies *version requirement* ...
Return 1 if *version* satisfies at least one of *requirement*s, 0 otherwise. Requirements are in any of the following forms (where *min* and *max* are valid version numbers; version must be greater than or equal to *min*):

min	Min-bounded (must be less than next major version).
min−	Min-unbound.
min−*max*	Bounded (must be less than *max*).

::pkg::create -name *pkgName* **-version** *pkgVersion* [**-load** *filespec*] ... [**-source** *filespec*] ...
Provided through Tcl library. Construct an appropriate **package ifneeded** command for a given package specification. At least one **-load** or **-source** parameter must be given.

pkg_mkIndex [**-direct**] [**-lazy**] [**-load** *pkgPat*] [**-verbose**] [--] *dir* [*pattern* ...]
Provided through Tcl library. Create index files that allow packages to be loaded automatically when **package require** commands are executed. See **pkg_mkIndex**(n) manual page for more information.

Package version numbers consist of one or more decimal numbers separated by dots, such as 2 or 1.162 or 3.1.13.1. The first number is called the major version number. Larger numbers correspond to later versions of a package. In addition, the letters “a” (alpha) and/or “b” (beta) may appear exactly once to replace a dot for separation. These letters semantically add a negative specifier into the version, where “a” is -2, and “b” is -1. Thus 1.3a1 becomes (semantically) 1.3.-2.1 and 1.3b1 is 1.3.-1.1. A version number not containing the letters “a” or “b” as specified above is called a **stable** version, whereas presence of the letters causes the version to be called is **unstable**.

16. Namespaces

The **namespace** command allows creating, accessing and destroying separate contexts for commands and variables. Commands and variables of different namespaces will not interfere with each other. Tcl always has one “global namespace” (with the empty string as name). Namespaces can nest hierarchically, and commands and variables inside can be referred to directly with qualified names, using “`::`” as hierarchical separator (e.g. `::namesp1::namesp2::cmd` or `$::namesp1::namesp2::var`). Namespaces are created with the **namespace eval** subcommand.

Following subcommands (which may be abbreviated) are available:

namespace children [*namespace*] [*pattern*]
Return a list of child namespaces belonging to *namespace* (defaults to current) which match *pattern* (default *).

namespace code *script*
Return a new script string which when evaluated arranges for *script* to be evaluated in current namespace. Useful for callbacks.

namespace current
Return a fully-qualified name of current namespace.

namespace delete [*namespace* . . .]
Each given namespace is deleted along with their child namespaces, procedures, and variables.

namespace ensemble *subcommand* [*arg* . . .]
Creates and manipulates a command that is formed out of an ensemble of subcommands. The following subcommands are defined:

namespace ensemble create [*option value* . . .]
Create a new ensemble command linked to the current namespace and return the fully-qualified name of it.

namespace ensemble configure *command* [*option] [value* . . .]
Retrieve option values or update options associated with the given ensemble command.

namespace ensemble exists *command*
Return 1 if the given command exists and is an ensemble command, 0 otherwise.

The following options are supported by the **namespace ensemble** subcommands:

-command *name*
A write-only option allowing the name of the ensemble created by **namespace ensemble create** to be anything in any namespace. Default is the fully-qualified name of the namespace in which **namespace ensemble create** is invoked.

-map [*dict*]
With *dict*, supply a dictionary that provides a mapping from subcommand names to a list of prefix words to substitute in place of the ensemble command and subcommand words. Without *dict*, the mapping will be from the local name of the subcommand to its fully-qualified name.

-namespace
A read-only option to **namespace ensemble configure** allowing the retrieval of the fully-qualified name of the namespace which the ensemble was created within.

-parameters [*arg_list*]
Provide a list of named arguments that are passed by the caller of the ensemble between the name of the ensemble and the subcommand argument. An empty list by default.

-prefixes [*boolean*]
Control whether the ensemble command recognizes unambiguous prefixes of its subcommands (default). When turned off, the ensemble command requires exact matching of subcommand names.

-subcommands [*subcommand_list*]
With *subcommands*, the option lists exactly what subcommands are in the ensemble. Without *subcommands*, the subcommands of the namespace will either be the keys of the dictionary listed in the **-map** option or the exported commands of the linked namespace.

-unknown [*partial_command*]
With *partial_command*, provide a partial command to handle the case where an ensemble subcommand is not recognized and would otherwise generate an error. Without *partial_command*, an error is generated whenever the ensemble is unable to determine how to implement a particular subcommand (default).

namespace eval *namespace arg* [*arg* . . .]
Activates *namespace* and evaluates concatenation of *args*'s inside it.

namespace exists *namespace*
Return 1 if *namespace* is valid in the current context, 0 otherwise.

namespace export [-clear] [*pattern* . . .]
Add all commands that match the given glob-style *pattern*'s to the export list of the current namespace. If **-clear** is given, the export list is first emptied. Without arguments, return the namespace's current export list.

namespace forget [[*namespace*::]*pattern* . . .]
Remove previously imported commands from a namespace that match glob-style *pattern*. Each pattern can be prefixed by a qualified namespace name (e.g. `a::b::p*`).

namespace import [**-force**] [*namespace*::*pattern* . . .]
Import commands that match the given glob-style *pattern*'s from an exporting *namespace*. The `-force` option allows replacing of existing commands. Without arguments, return a list of commands in the current namespace that have been imported from other namespaces (without namespace qualifiers).

namespace inscope *namespace script* [*arg* . . .]
Execute *script* in the context of the specified *namespace*. Not expected to be used directly by programmers, and much like the **namespace eval** command except that *namespace* must already exist.

namespace origin *command*
Return a fully-qualified name of imported *command*.

namespace parent [*namespace*]
Return a fully-qualified name of parent namespace of *namespace* (defaulting to the current namespace).

namespace path [*namespaceList*]
Return or set the command resolution path of the current namespace.

namespace qualifiers *string*
Return any leading namespace qualifiers in *string*.

namespace tail *string*
Return the simple name at the end of *string* (strips namespace qualifiers).

namespace upvar *namespace* [*otherVar myVar* . . .]
Arrange for zero or more local variables in the current procedure to refer to variables in *namespace*.

namespace unknown [*script*]
Set or return the unknown command handler for the current namespace.

namespace which [-command | -variable] *name*
Return the fully-qualified name of the command (or the variable, if `-variable` is given) *name* in the current namespace. Will look in the global namespace if not found in the current namespace.

variable [*name value* . . .] *name* [*value*]
Create one or more variables in the current namespace (if *name* is unqualified), optionally initialized to the given *value*s. Inside a procedure, a local variable is created linked to the specified namespace variable.

17. Multiple Interpreters

The **interp** command is used to create, delete, and manipulate child interpreters, and to share or transfer channels between interpreters. Different interpreters are independent from each other and have their own name spaces. A qualified interpreter name is a proper Tcl list containing a subset of its ancestors in the interpreter hierarchy. For example, if "**a**" is a child of the current interpreter and it has a child "**a1**", which in turn has a child

"**a11**", the qualified name of "**a11**" in "**a**" is the list "**a1 a11**". The current interpreter can always be referred to as "**{ }**" (empty list or string). In the below qualified interpreter names are referred to as *path*s.

Following subcommands are available:

interp alias *srcPath srcToken*
: Return a list whose elements are the *targetCmd* and *args* associated with the alias *srcToken* in interpreter *srcPath*.

interp alias *srcPath srcToken* **{ }**
: Delete the alias *srcToken* in interpreter *srcPath*.

interp alias *srcPath srcCmd targetPath targetCmd* [*arg* . . .]
: Create an alias *srcCmd* in interpreter *srcPath* which when invoked will run *targetCmd* and *arg*s in the interpreter *targetPath*.

interp aliases [*path*]
: Return a list of all aliases defined in interpreter *path*.

interp bgerror *path* [*cmdPrefix*]
: Get or set the current background error handler for interpreter *path*.

interp cancel [**-unwind**] [--] [*path*] [*result*]
: Cancel the script being evaluated in interpreter *path*. With the **`-unwind`** option the evaluation stack for the interpreter is unwound without regard to any intervening catch command until there are no further invocations of the interpreter left on the call stack. If *result* is present, it will be used as the error message string.

interp children [*path*]
: A synonym fo **interp slaves**.

interp create [**-safe**] [--] [*path*]
: Create a slave interpreter identified by *path* and a new command *child command*. The name of the child command is the last component of *path*. Without *path*, a unique name "**interp***x*" is created, where *x* is an integer. An interpreter with limited functionality can be created with the **`-safe`** option (see safe interpreters below). The result of the command is the name of the new interpreter.

interp debug *path* [**-frame** [*bool*]]
: Control whether to capture frame-level stack information in slave interpreter *path*. without arguments, return option and current setting. If **`-frame`** is given, the debug setting is set to the given boolean if provided and the current setting is returned.

interp delete [*path* . . .]
: Delete the interpreter(s) *path* and all its child interpreters.

interp eval *path arg* [*arg* . . .]
: Evalute concatenation of *arg*s as command in interpreter *path*. Return the evaluation result to the invoking interpreter.

interp exists [*path*]
: Return 1 if interpreter *path* exists, 0 otherwise.

interp expose *path hiddenName* [*exposedCmdName*]
: Make hidden command *hiddenName* in interpreter *path* exposed (optionally as *exposedCmd*).

interp hide *path exposedCmdName* [*hiddenCmdName*]
: Make exposed command *exposedCmdName* in interpreter *path* hidden (optionally as *hiddenCmdName*).

interp hidden *path*
: Return a list of hidden commands in interpreter *path*.

interp invokehidden *path* [*option* . . .] [--] *hiddenCmdName* [*arg* . . .]
Invoke hidden command *hiddenCmdName* with specified *arg*s in interpreter *path*. Supported options are:

-global Invoke at global level.

-namespace *nsName* Invoke in namespace *nsName*.

interp issafe [*path*]
Return 1 if interpreter *path* is safe (see safe interpreters below), 0 otherwise.

interp limit *path limitType* [*option*] [*value* . . .]
Set up, manipulate and query the configuration of resource limit *limitType* for interpreter *path*. Possible *limitType*s are **commands** and **time**. When a limit is exceeded, an error is generated after any handler callbacks defined by parent interpreters are called. Supported limit options are:

-command [*script*]
For all limit types, specify (or query) a Tcl script (command) to be executed in the global namespace of the interpreter reading and writing the option when the particular limit in the limited interpreter is exceeded.

-granularity [*integer*]
For all limit types, specify (or query) an integer divisor, which must be at least 1 and which indicates how frequently the limit is to be checked.

-milliseconds [*ms*]
Specify (or query) the number of milliseconds after the moment defined in the **-seconds** option that the time limit will fire.

-seconds [*s*]
Specify (or query) the number of seconds after the epoch (see **clock seconds**) that the time limit for the interpreter will be triggered. An empty string can be specified to indicate that a time limit is not set for the interpreter.

-value [*nr*]
Specify (or query) the number of commands that the interpreter may execute before triggering the command limit. An empty string can be specified to indicate that a command limit is not set for the interpreter.

interp marktrusted [*path*]
Mark interpreter *path* as trusted. Any hidden commands will not be exposed.

interp recursionlimit *path* [*newLimit*]
Return or set the maximum allowable nesting depth for interpreter *path*.

interp share *srcPath channelId destPath*
Arrange for I/O channel *channelId* in interpreter *srcPath* to be shared with interpreter *destPath*. Both interpreters must close it to close the underlying IO channel.

interp slaves [*path*]
Return a list of names of all slave interpreters of interpreter *path*.

interp target *path alias*
Return a list describing the target interpreter of *alias* in interpreter *path*.

interp transfer *srcPath channelId destPath*
Move I/O channel *channelId* from interpreter *srcPath* to *destPath*.

Child Commands

For each child interpreter created with the **interp** command, a new Tcl command is created in the parent interpreter with the same name as the new interpreter. This command may be used to invoke various operations on the interpreter. Following child commands are available (see above for explanations):

child **alias** *srcToken*
child **alias** *srcToken* **{ }**
child **alias** *srcCmd targetCmd* [*arg* . . .]
child **aliases**
child **bgerror** [*cmdPrefix*]
child **eval** *arg* [*arg* . . .]
child **expose** *hiddenName* [*exposedCmdName*]
child **hide** *exposedCmdName* [*hiddenCmdName*]
child **hidden**
child **invokehidden** [*option* . . .] [--] *hiddenCmdName* [*arg* . . .]
child **issafe**
child **limit** *limitType* [*option*] [*value* . . .]
child **marktrusted**
child **recursionlimit** [*newLimit*]

Safe Interpreters

A safe interpreter is one with restricted functionality, so that it is safe to execute an arbitrary script without damaging the enclosing application or computing environment. Certain commands and variables are removed from the safe interpreter. Limited access to these facilities can be provided, by creating aliases to the parent interpreter and restricting capabilities here.

A safe interpreter is created with exactly the following set of built-in commands:

after	**error**	**info**	**lsort**	**split**
append	**eval**	**interp**	**namespace**	**string**
apply	**expr**	**join**	**package**	**subst**
array	**fblocked**	**lappend**	**pid**	**switch**
binary	**fcopy**	**lassign**	**proc**	**tell**
break	**fileevent**	**lindex**	**puts**	**time**
catch	**flush**	**linsert**	**read**	**trace**
chan	**for**	**list**	**regexp**	**unset**
clock	**foreach**	**llength**	**regsub**	**update**
close	**format**	**lrange**	**rename**	**uplevel**
concat	**gets**	**lrepeat**	**return**	**upvar**
continue	**global**	**lreplace**	**scan**	**variable**
dict	**if**	**lsearch**	**seek**	**vwait**
eof	**incr**	**lset**	**set**	**while**

The following commands are hidden within a safe interpreter, and can be recreated later as Tcl procedures or aliases, or re-exposed with **interp expose**:

cd	**exit**	**glob**	**pwd**	**source**
encoding	**fconfigure**	**load**	**socket**	**unload**
exec	**file**	**open**		

Safe Tcl

Safe Tcl is a mechanism for executing untrusted Tcl scripts safely and for providing mediated access by such scripts to potentially dangerous functionality. Safe Tcl allows a parent interpreter to create safe, restricted interpreters that contain a set of predefined aliases for the **source**, **load**, **file**, **encoding**, and **exit** commands and are able to use the auto-loading and package mechanisms. No knowledge of the file system structure is leaked to the safe interpreter, because it has access only to a virtualized path containing tokens. All commands provided in the parent interpreter by Safe Tcl reside in the **safe** namespace. See the **safe**(n) manual page for more information.

The following commands are available:

::safe::interpCreate [*child*] [*options* . . .]
Create a safe interpreter, install the predefined aliases (see below) and initialize the auto-loading and package mechanism as specified by the supplied *options* (see below). A name will be generated if *child* is not specified. Returns the interpreter name.

::safe::interpInit *child* [*options* . . .]
Like **::safe::interpCreate**, except that *child* must have been created before (e.g. with **interp create -safe**).

::safe::interpConfigure *child* [*options* . . .]
Without *options*, return a list with the settings for all options of the named safe interpreter *child*. With a single argument, return a 2 element list with the full name of the specified option and its value for *child*. With more than two arguments, reconfigure the the safe interpreter *child* as per specified *options* (see below).

::safe::interpDelete *child*
Deletes the safe interpreter and cleans up the corresponding parent interpreter data structures. If a deleteHook script (see below was specified for this interpreter it is evaluated before the interpreter is deleted, with the name of the interpreter as an additional argument.

::safe::interpAddToAccessPath *child directory*
Add *directory* the virtual path maintained for the safe interpreter in the parent, and returns the token that can be used in the safe interpreter to obtain access to files in that directory. If the directory is already in the virtual path, it only returns the token without adding the directory to the virtual path again.

::safe::interpFindInAccessPath *child directory*
Find and return the token for the real *directory* directory in the safe interpreter's current virtual access path. It generates an error if the directory is not found.

::safe::setLogCmd [*cmd arg* . . .]
Install a script that will be called when interesting life cycle events occur for a safe interpreter. Without *cmd*, return the currently installed script. With *cmd* being an empty string and only argument, the currently installed script is removed and logging is turned off. The script will be invoked with one additional argument, a string describing the event of interest.

The following options are common to the above ::safe::interpCreate, ::safe::interpInit, and ::safe::interpConfigure commands and can be abbreviated as long as non-ambiguous:

-accessPath *directoryList*
Set the list of directories from which the safe interpreter can source and load files. Default is to use the same directories as the parent for auto-loading.

-statics *boolean*
Specify if the safe interpreter will be allowed to load statically linked packages. Default is **true**.

-noStatics
A shortcut for **-statics false**.

-nested *boolean*
Specify if the safe interpreter will be allowed to load packages into its own sub-interpreters. Default is **false**.

-nestedLoadOk
A shortcut for **-nested true**.

-deleteHook *script*
The specified script will be evaluated in the parent with the name of the safe interpreter as an additional argument just before deleting the safe interpreter.

Specifying an empty value will remove any currently installed deletion hook script. Default is not to have any deletion call back.

The following aliases are provided in a safe interpreter:

source *fileName*
Files can only be sourced from directories in the virtual path for the safe interpreter and requires the use of a token name.

load *fileName*
Shared object files can only be loaded from directories in the virtual path for the safe interpreter and requires the use of a token name.

file [*subCmd args . . .*]
Only the following subcommands are available: **dirname**, **join**, **extension**, **root**, **tail**, **pathname** and **split**.

encoding [*subCmd args . . .*]
Setting the system encoding is disallowed.

exit The calling interpreter is deleted and its computation is stopped, but the Tcl process in which this interpreter exists is not terminated.

18. Coroutines

Commands to create and produce values from coroutines. See the **coroutine**(n) manual page for more information and examples.

coroutine *name command* [*arg. . .*]
Create a new coroutine context (with associated command) named *name* and executes that context by calling *command*, passing in the other remaining arguments without further interpretation. Once *command* returns normally or with an exception (e.g., an error) the coroutine context *name* is deleted.

yield [*value*]
Within the context, values may be generated as results by using the **yield** command; if no value is supplied, the empty string is used. The context will suspend execution and the **coroutine** command will return the argument to **yield**.

yieldto *command* [*arg. . .*]
The coroutine may also suspend its execution by use of the **yieldto** command, which instead of returning, cedes execution to some command called *command* (resolved in the context of the coroutine) and to which any number of arguments may be passed.

name [*value. . .*]
The coroutine that can be executed.

19. HTTP/1.1 Protocol

See the **http**(n) manual page for more information on following commands providing the client side of the HTTP/1.1 protocol:

package require http [**2.9**]
::http::config [*-option value . . .*]
::http::geturl *url* [*-option value . . .*]
::http::formatQuery *key value* [*key value . . .*]
::http::quoteString *value*
::http::reset *token* [*why*]
::http::wait *token*

::http::status *token*
::http::size *token*
::http::code *token*
::http::ncode *token*
::http::meta *token*
::http::data *token*
::http::error *token*
::http::cleanup *token*
::http::register *proto port command*
::http::registerError *port* [*message*]
::http::unregister *proto*

20. Object Oriented Tcl

See following manual pages for more information on object oriented extensions to Tcl:

my(n)	Invoke any method of current object.
next(n)	Invoke superclass method implementations.
nextto(n)	Invoke superclass method implementations.
oo::class(n)	Class of all classes.
oo::copy(n)	Create copies of objects and classes.
oo::define(n)	Define and configure classes and objects.
oo::objdefine(n)	Define and configure classes and objects.
oo::object(n)	Root class of the class hierarchy.
self(n)	Method call internal introspection.

Class Introspection

The following subcommand values are supported by **info class**:

info class call *class method*
: Return a list of lists of four elements as description of the method implementations that are used to provide a stereotypical instance of *class*'s implementation of *method*.

info class constructor *class*
: Return a two element list as description of the definition of the constructor of class *class*.

info class definition *class method*
: Return a two element list as description of the definition of the method named *method* of class *class*.

info class destructor *class*
: Return the body of the destructor of class *class*.

info class filters *class*
: Return the list of filter methods set on class *class*.

info class forward *class method*
: Return the argument list for the method forwarding called *method* that is set on the class called *class*.

info class instances *class* [*pattern*]
: Return a list of instances of class *class*, optionally string matching *pattern*.

info class methods *class* [*option* . . .]
: Return a list of all public (i.e. exported) methods of the class called *class*. Possible options are **`-all`** and **`-private`**.

info class methodtype *class method*
Return a description of the type of implementation used for the method named *method* of class *class*. The result can be **method** or **forward**.

info class mixins *class*
Return a list of all classes that have been mixed into the class named *class*.

info class subclasses *class* [*pattern*]
Return a list of direct subclasses of class *class*, optionally string matching *pattern*.

info class superclasses *class*
Return a list of direct superclasses of class *class* in inheritance precedence order.

info class variables *class*
Return a list of all variables that have been declared for the class named *class*.

Object Introspection

The following subcommand values are supported by **info object**:

info object call *object method*
Return a list of lists of four elements as description of the method implementations that are used to provide *object*'s implementation of *method*.

info object class *object* [*className*]
Return a boolean value indicating whether the *object* is of given class or, if *className* is not specified, the class of the object *object*.

info object definition *object method*
Return a two element list as description of the definition of the method named *method* of object *object*.

info object filters *object*
Return the list of filter methods set on *object*.

info object forward *object method*
Return the argument list for the method forwarding called *method* that is set on the object called *object*.

info object isa *category object* [*arg*]
Return a boolean value that indicates whether the *object* argument meets the criteria for the *category*. The supported categories are:

info object isa class *object* [*arg*]
Return whether *object* is a class.

info object isa metaclass *object* [*arg*]
Return whether *object* is a class that can manufacture classes.

info object isa mixin *object* [*arg*]
Return whether *class* is directly mixed into *object*.

info object isa object *object* [*arg*]
Return whether *object* really is an object.

info object isa typeof *object* [*arg*]
Return whether *class* is the type of *object*.

info object methods *object* [*option* . . .]
Return a list of all public (i.e. exported) methods of the object called *object*. Possible options are **-all** and **-private**.

info object methodtype *object method*
Return a description of the type of implementation used for the method named *method* of object *object*. The result can be **method** or **forward**.

info object mixins *object*
Return a list of all classes that have been mixed into the object named *object*.

info object namespace *object*
Return the name of the internal namespace of the object named *object*.

info object variables *object*
Return a list of all variables that have been declared for the object named *object*.

info object vars *object* [*pattern*]
Return a list of all variables in the private namespace of the object named *object*, optionally string matching *pattern*.

21. Other Tcl Commands

A variety of commands not fitting into the categories covered by the previous chapters.

after *ms* [*script script script* . . .]
Arrange for command (concat of *script* arguments) to be run after *ms* milliseconds have passed. Without *script* arguments, the program will sleep for *ms* milliseconds. With the *script* arguments, returns an identifier that can be used to cancel the delayed command using **after cancel**.

after cancel *id* | *script script* . . .
Cancels the execution of a delayed command that was previously scheduled. Either by specifying the id returned from a previous **after** command, or by specifying the name of a pending command specified to a previous **after** command.

after idle *script* [*script script* . . .]
Arrange for command (concat of *script*) to be evaluated later as an idle callback. The script will be run exactly once, the next time the (Tk) event loop is entered and there are no events to process. Returns an identifier that can be used to cancel the delayed command using **after cancel**.

after info [*id*]
Return information on event handler *id*. Without *id*, return a list of all existing event handler ids.

apply *func* [*arg* . . .]
Apply function *funct* to the given arguments and return the result.

auto_execok *cmd*
Provided through Tcl library. Return a list of arguments to be passed to **exec** if an executable file or shell builtin by the name *cmd* exists in user's **PATH**, empty string otherwise.

auto_import *pattern*
Provided through Tcl library. Invoked during **namespace import** to see if imported commands specified by *pattern* reside in an autoloaded library.

auto_load *cmd*
Provided through Tcl library. Attempts to load the definition for *cmd* by searching the variable **$auto_path** or **$env(TCLLIBPATH)** for a **tclIndex** file, which will inform the interpreter where it can find *cmd*'s definition. Returns 1 if *cmd* was successfully created, 0 otherwise.

auto_mkindex *dir* [*pattern* . . .]
Provided through Tcl library. Generate a **tclIndex** file from all files in *dir* that match glob *patterns* (defaulting to ***.tcl**).

auto_qualify *command namespace*
Provided through Tcl library. Compute a list of fully qualified names for *command*.

auto_reset
Provided through Tcl library. Destroy cached information used by **auto_execok** and **auto_load**.

catch *script* [*resultVarName*] [*optionsVarName*]
Evaluate *script* without raising errors and optionally store results into *resultVarName*. Optionally store a directory of return options into *optionsVarName*. If there is an error, a non-zero error code is returned and an error message stored in *resultVarName*.

dde *subcommand args*
Execute a Dynamic Data Exchange (DDE) command when running under Microsoft Windows. See **dde**(n) manual page for details.

error *message* [*info*] [*code*]
Interrupt command interpretation with an error described in *message*. The **-errorinfo** and **-errorcode** return options can be set to *info* and *code*.

eval *arg* [*arg* . . .]
Return the result of evaluating the concatenation of *args*'s as a Tcl command.

expr *arg* [*arg* . . .]
Return the result of evaluating the concatenation of *arg*'s as an operator expression. See Operators and Expressions for more info.

global *varName* [*varName* . . .]
Declares given *varName*'s as global variables within a **proc** body.

history *subcommand* [*arg* . . .]
Manipulate the command history list. See **history**(n) manual page for details.

incr *varName* [*increment*]
Increment the integer value stored in *varName* by *increment* (default 1) and return the result. If *varName* is unset, set it to *increment* or to 1 by default.

load [**-global**] [**-lazy**] [--] *fileName* [*prefix* [*interp*]]
Load binary code from a file *fileName* into the application's address space and call an initialization procedure in the library to incorporate it into an interpreter. If specified, *prefix* is used to compute the name of an initialization procedure, and *interp* can be specified as path name of the interpreter into which to load the library. With **-global**, all symbols found in the shared library are exported for global use by other libraries. With **-lazy**, the actual loading of symbols is delayed until their first actual use.

memory *command* [*arg* . . .]
Control Tcl memory debugging capabilities. Only available when Tcl has been compiled with memory debugging enabled. See **memory**(n) manual page for details.

::msgcat::*command* [*arg* . . .]
The **msgcat** package provides a set of functions that can be used to manage multi-lingual user interfaces using an application independent "message catalog". See **msgcat**(n) manual page for details.

platform::*command* [*arg*]
The **platform** package provides several utility commands useful for the identification of the architecture of a machine running Tcl. Commands available are: **generic**, **identify**, and **patterns**. See **platform**(n) manual page for details.

platform::shell::*command shell*
The **platform:shell** package provides several utility commands useful for the identification of the architecture of a specific Tcl shell. Commands available are: **generic**, **identify**, and **platform**. See **platform::shell**(n) manual page for details.

proc *name args body*
Create a new Tcl procedure (or replace existing one) called *name* where *args* is a list of arguments and *body* Tcl commands to evaluate when invoked. *Args* can be an empty list (no arguments) or a list of 2-element lists. This 2-element list

specifies the argument name and its default value. A local variable is created for each of the formal arguments to the procedure; its value will be the value of corresponding argument in the invoking command or the argument's default value. If the last argument has the name **args**, then this will be a list containing the values of any remaining arguments when invoked. Examples:

```
proc mult {varName {multiplier 2}} {
    upvar 1 $varName var
    set var [expr {$var * $multiplier}]
}
proc printArguments args {
    foreach arg $args {
        puts $arg
    }
}
```

registry [*-mode*] *command keyName arg* . . .
Manipulate the Microsoft Windows registry. See **registry**(n) manual page for details.

rename *oldName newName*
Rename command *oldName* so it is now called *newName*. If *newName* is the empty string, command *oldName* is deleted.

set *varName* [*value*]
Store *value* in *varName* if given. Returns the current value of *varName*. *VarName* can specify a an array element.

source [**-encoding** *encoding*] *fileName*
Read file *fileName* and evaluate its contents as a Tcl *script*. The encoding of *fileName* can be specified. The return value from **source** is the return value of the last command executed in the script. If a **return** is invoked from within the script then the remainder of the file will be skipped and the **source** command will return normally with the result from the **return** command.

tailcall *command* [*arg* . . .]
Replace the current procedure with another command. This is equivalent to: **return [uplevel 1 [list** *command arg* . . . **]]**

tcl_findLibrary *basename version patch initScript enVarName varName*
Provided through Tcl library. A standard search procedure for use by extensions during their initialization.

tcltest::*command arg* . . .
The **tcltest** package provides several utility commands useful in the construction of test suites for code instrumented to be run by evaluation of Tcl commands. See **tcltest**(n) manual page for details.

::tcl::tm::*command arg* . . .
Facilities for locating and loading of Tcl Modules. See **tm**(n) manual page for details.

throw *type message*
This command causes the current evaluation to be unwound with an error. The error created is described by the *type* and *message* arguments: *type* must contain a list of words describing the error in a form that is machine-readable (and which will form the error-code part of the result dictionary), and *message* should contain text that is intended for display to a human being.

time *script* [*count*]
Call interpreter *count* (default 1) times to evaluate *script*. Returns a string of the form "`503.2 microseconds per iteration`". Time is measured in elapsed time, not CPU time.

timerate [*options args*] *script* [*time*] [*max-count*]
Calibrated performance measurements of script execution time. See **timerate**(n)

manual page for details.

trace add | remove | info *type name* [*ops commandPrefix* [*arg* . . .]]
Add, remove or provide information on monitoring of operations specified with *type* (and further arguments): **command** for command renaming or deletion, **execution** for command execution, and **variable** for variable access. See **trace**(n) manual page for details.

try *body* [*handler* . . .] [**finally** *script*]
Trap and process errors and exceptions. Execute the script *body* and, depending on what the outcome of that script is (normal exit, error, or some other exceptional result), run a *handler* script to deal with the case. Once that has all happened, if the **finally** clause is present, the *script* it includes will be run and the result of the handler (or the *body* if no handler matched) is allowed to continue to propagate.

The *handler* clauses are each expressed as several words, and must have one of the following forms:

on *code variableList script*
Matches if the evaluation of *body* completed with the exception code *code*. *Code* can be **ok**, **error**, **return**, **break**, or **continue** (or the equivalent integers 0 through 4).

trap *pattern variableList script*
Matches if the evaluation of *body* resulted in an error and the prefix of the **-errorcode** from the interpreter's status dictionary is equal to the *pattern*.

If *variableList* is non-empty, the first variable name will contain the result of the evaluation of *body*. Any second variable name will contain the options dictionary of the interpreter at the moment of completion of execution of *body*.

The *script* of each *handler* is a Tcl script to evaluate if the clause is matched. If *script* is a literal "**–**" and the handler is not the last one, the *script* of the following handler is invoked instead (as with the **switch** command).

unknown *cmdName* [*arg* . . .]
Called when the Tcl interpreter encounters an undefined command name.

unload [**-nocomplain**] [**-keeplibrary**] [--] *fileName* [*prefix* [*interp*]]
Try to unload shared libraries previously loaded with **load**. Where *fileName* is the name of the file containing the library file to unload, which must be the same as used with **load**. The *prefix* argument is the prefix (as determined by or passed to **load**), and is used to compute the name of the unload procedure. The *interp* argument is the path name of the interpreter from which to unload the package.

With **-nocomplain** all error messages can be suppressed. With **-keeplibrary** the operating system will not unload the library from the process.

unset [**-nocomplain**] [--] *name* [*name* . . .]
Removes the given variables, arrays and array elements from scope. Possible errors can be suppressed with **-nocomplain**.

update [**idletasks**]
Handle pending (Tk) events. If **idletasks** is specified, only those operations normally deferred until the idle state are processed.

uplevel [*level*] *arg* [*arg* . . .]
Evaluates concatenation of *arg*'s in the variable context indicated by *level*, an integer (defaulting to 1) that gives the distance up the calling stack. If *level* is preceded by "**#**", then it gives the distance down the calling stack from the global level. The result of the evaluation is returned.

upvar [*level*] *otherVar myVar* [*otherVar myVar* . . .]
Makes *myVar* in local scope equivalent to *otherVar* at context *level* (see **uplevel**) so they share the same storage space.

vwait *varName*

Enter Tcl event loop until global variable *varName* is modified. A fully-qualified variable name name can be used to refer to other namespaces.

22. Pattern Globbing

Several Tcl commands support file name or argument pattern matching by "globbing" in a fashion similar to the csh or bash shell. Following "globbing" patterns are supported:

`?`	Match any single character.
`*`	Match zero or more characters.
`[`*chars*`]`	Match set of characters.
`[`*a*`-`*z*`]`	Match range of characters.
x	Match character *x*.
`{`*a*`,`*b*`,...}`	Match any of strings *a*, *b*, etc.
~	Home directory (for **glob** command).
~*user*	Match *user*'s home directory (for **glob** command).

Note: For the **glob** command, a "`.`" at the beginning of a file's name or just after "`/`" and all "`/`" characters must be matched explicitly.

See **glob**(n) manual page for details.

23. Regular Expressions

An advanced regular expression ("ARE") is one or more *branches*, separated by "`|`", matching anything that matches any of the branches. A branch is zero or more *constraints* or *quantified atoms*, concatenated. It matches a match for the first, followed by a match for the second, etc; an empty branch matches the empty string.

See **re_syntax**(n) manual page for details.

Quantifiers

A quantified atom is an *atom* possibly followed by a single *quantifier*. Without a quantifier, it matches a single match for the atom.

re`*`	Match zero or more of *re*.
re`+`	Match one or more of *re*.
re`?`	Match zero or one of *re*.
re`{`*m*`}`	Match *re* exactly *m* times.
re`{`*m*`,}`	Match *re* at least *m* times.
re`{`*m*`,`*n*`}`	Match *re* at least *m* and at most *n* times.
`*? +? ?? {`*m*`}? {`*m*`,}? {`*m*`,`*n*`}?`	"Non-greedy" quantifiers, preferring the smallest instead of the largest number of matches.

Atoms

`(`*re*`)`	Matches a match for *regexp* with the match noted for possible reporting.
`(?:`*re*`)`	As previous, but does no reporting.
`()`	Matches an empty string, noted for possible reporting.
`(?:)`	Matches an empty string, without reporting.
`[`*chars*`]`	A bracket expression, matching any one of the chars (see below).
`.`	Any single character except newline.

\k	Match non-alphanumeric character *k* taken as an ordinary character, e.g. \\ matches a backslash character.
\c	Where *c* is alphanumeric, an *escape* (see below).
{	When followed by a character other than a digit, matches the left-brace character "**{**"; when followed by a digit, it is the beginning of a bound quantifier (see above).
x	Where *x* is a single character with no other significance, matches that character.

Constraints

A *constraint* matches an empty string when specific conditions are met. A constraint may not be followed by a quantifier.

^	Match at the beginning of a line.
$	Match at the end of a line.
(?=*re***)**	Positive lookahead, matches at any point where a substring matching *re* begins.
(?!*re***)**	Negative lookahead, matches at any point where no substring matching *re* begins.

Bracket Expressions

[*chars***]**	Match characters in set.
[^*chars***]**	Match characters not in set.
[*a***-***z***]**	Match range of characters.
[^*a***-***z***]**	Match characters not in range.

Character Classes

Within a bracket expression, the name of a character class enclosed in **[:** and **:]** stands for the list of all characters (not all collating elements!) belonging to that class.

alpha	A letter.
upper	An upper-case letter.
lower	A lower-case letter.
digit	A decimal digit.
xdigit	A hexadecimal digit.
alnum	An alphanumeric (letter or digit).
print	A "printable" (same as graph, except also including space).
blank	A space or tab character.
space	A character producing white space in displayed text.
punct	A punctuation character.
graph	A character with a visible representation (includes both **alnum** and **punct**).
cntrl	A control character.

Collating Elements

Within a bracket expression, a collating element (a character, a multi-character sequence that collates as if it were a single character, or a collating-sequence name for either) enclosed in **[.** and **.]** stands for the sequence of characters of that collating element. For example, the RE "**[[.ch.]]*c**" (zero or more "**ch**s" followed by "c") matches the first five characters of "**chchcc**".

Escapes

Escapes , which begin with a **** followed by an alphanumeric character, come in several varieties: character entry, class shorthands, constraint escapes, and back references.

Character-entry Escapes

a	Alert (bell) character.
b	Backspace.
B	Synonym for **** to help reduce backslash doubling.
c*X*	(where *X* is any character) the character whose low-order 5 bits are the same as those of *X*, and whose other bits are all zero (a control character).
e	Escape character (the character with octal value 033).
f	Formfeed.
n	Newline.
r	Carriage return.
t	Horizontal tab.
u*hhhh*	Where *hhhh* is one up to four hexadecimal digits, the Unicode character **U+***hhhh* in the local byte ordering.
U*hhhhhhhh*	Where *hhhhhhhh* is one up to eight hexadecimal digits, reserved for a Unicode extension up to 21 bits. The digits are parsed until the first non-hexadecimal character is encountered, the maximun of eight hexadecimal digits are reached, or an overflow would occur in the maximum value of **U+**10ffff.
v	Vertical tab.
x*hh*	Where *hh* is one or two hexadecimal digits, the character whose hexadecimal value is **0x***hh*.
0	The character whose value is **0**.
xyz	Where *xyz* is exactly three octal digits, and is not a *back reference*, the character whose octal value is **0***xyz*. The first digit must be in the range 0-3, otherwise the two-digit form is assumed.
xy	Where *xy* is exactly two octal digits, and is not a back reference, the character whose octal value is **0***xy*.

Hexadecimal digits are "**0**-**9**", "**a**-**f**", and "**A**-**F**". Octal digits are "**0**-**7**".

Class-shorthand Escapes

d **D**	**[[:digit:]]** and **[^[:digit:]]**.
s **S**	**[[:space:]]** and **[^[:space:]]**.
w **W**	**[[:alnum:]_]** and **[^[:alnum:]_]** (note the underscore).

Constraint Escapes

A	Matches only at the beginning of the string.
m	Matches only at the beginning of a word.
M	Matches only at the end of a word.
y	Matches only at the beginning or end of a word.
Y	Matches only at a point that is not the beginning or end of a word.
Z	Matches only at the end of the string.
m	Where *m* is a nonzero digit, a *back reference* (see below).
mnn	Where *m* is a nonzero digit and *nn* is some more digits, and the decimal value *mnn* is not greater than the number of closing capturing parentheses seen so far, a back reference (see below).

A word is defined as a sequence of word characters that is neither preceded nor followed by word characters. A word character is an *alnum* character or an underscore ("_").

Back References

A back reference matches the same string matched by the parenthesized subexpression specified by the number, so that (e.g.) "`([bc])\1`" matches "`bb`" or "`cc`" but not "`bc`". The subexpression must entirely precede the back reference in the RE. Subexpressions are numbered in the order of their leading parentheses.

Command Index

after, 43
append, 7
apply, 43
array, 15
auto_execok, 43
auto_import, 43
auto_load, 43
auto_mkindex, 43
auto_qualify, 43
auto_reset, 43

binary, 7
break, 4

catch, 44
cd, 18
chan, 26
clock, 18
close, 23
concat, 12
continue, 4
coroutine, 40

dde, 44
dict, 16

encoding, 8
eof, 23
error, 44
eval, 44
exec, 19
exit, 4
expr, 44

fblocked, 23
fconfigure, 23
fcopy, 24
file, 20
fileevent, 24
flush, 24
for, 4
foreach, 4
format, 8

gets, 24
glob, 20
global, 44

history, 44
http, 40

if, 4
incr, 44
info, 5
info class, 41
info object, 42
interp, 35

join, 13

lappend, 13
lassign, 13
lindex, 13
linsert, 13
list, 13
llength, 13
lmap, 13
load, 44
lrange, 13
lrepeat, 13
lreplace, 13
lreverse, 13
lsearch, 13
lset, 14
lsort, 14

memory, 44
msgcat, 44
my, 41

namespace, 33
next, 41
nextto, 41

oo::class, 41
oo::copy, 41
oo::define, 41
oo::objdefine, 41
oo::object, 41
open, 24

package, 32
parray, 16

pid, 20
pkg::create, 33
pkg_mkIndex, 33
platform, 44
platform::shell, 44
proc, 44
puts, 25
pwd, 20

read, 25
regexp, 8
registry, 45
regsub, 9
rename, 45
return, 5

safe, 38
scan, 9
seek, 25
self, 41
set, 45
socket, 25
source, 45
split, 15
string, 10
subst, 12
switch, 5

tailcall, 45
tcl::prefix, 12
tcltest, 45
tcl_endOfWord, 12
tcl_findLibrary, 45
tcl_startOfNextWord, 12
tcl_startOfPreviousWord, 12
tcl_wordBreakAfter, 12
tcl_wordBreakBefore, 12
tell, 25
throw, 45
time, 45
timerate, 45
tm, 45
trace, 46
try, 46

unknown, 46
unload, 46
unset, 46
update, 46
uplevel, 46
upvar, 46
variable, 35
vwait, 47
while, 5
yield, 40
yieldto, 40
zlib, 28

Notes

