TCL/TK Command Reference Guide

Applicable Versions:

	TCL	TK
Starting	8.0p2	8.0p2
Latest Addition	8.4.9/8.5a1	8.4.9/8.5a1

Contents

Fundamentals

Sect #	Section Title
1.1	Shells
1.2	System Variables
1.3	Syntax
1.4	Operators and Expressions
1.5	Pattern Globbing
1.6	Regular Expressions
1.7	Advanced Regular Expressions

TCL

Sect #	Section Title
2.1	Arrays
2.2	Clock
2.3	Command Evaluation
2.4	Control Loops
2.5	Dictionary
2.6	Encodings
2.7	Event Loop Handlers
2.8	File Attributes
2.9	History
2.10	Input/Output
2.11	Interpreter Information
2.12	Interpreters
2.13	Lists
2.14	Namespaces
2.15	Packages
2.16	Procedures
2.17	Strings
2.18	Variables

TK

Sect #	Section Title
3.1	Bindings and Events
3.2	Button Widget
3.3	Canvas Widget
3.4	Checkbutton
3.5	Clipboard and Selection
3.6	Console
3.7	Dialogs
3.8	Entry Widget
3.9	Fonts
3.10	Frame Widget
3.11	Geometry Management
3.12	Images
3.13	Label Widget
3.14	Labelframe Widget
3.15	Listbox Widget
3.16	Menu Widget
3.17	Menubutton Widget
3.18	Message Widget
3.19	Options and Resources
3.20	Panedwindow
3.21	Radiobutton Widget
3.22	Scale Widget
3.23	Scrollbar
3.24	Spinbox
3.25	Text Widget
3.26	Toplevel Window
3.27	Window Information
3.28	Window Management

Other TCL Packages

Sect #	Package
4.1	dde
4.2	http
4.3	msgcat
4.4	registry
4.5	resource
4.6	tcltest

Index

A	Command Index

Need to add console (8.3.4+ bindings)
Add packages http (8.0+), msgcat (8.1), opt (8.1), resource, tcltest (8.2+)
Finish 8.5 additions: dict

References:

1. Tcl/Tk v8.0p2 to 8.5 man pages
2. Tcl/Tk v8.0p2 to 8.5 source code
3. Changes in Tcl/Tk (http://mini.net/tcl/405)
4. Trial and error

Conventions

bold	Denotes literal text such as commands and option switches.
italic	Denotes variable text such as files, variables, etc. Generally *variable* refers to the variable contents while *variableName* refers to the name of the variable.
?. . . ?	Denotes an optional specifier.
<char>	Denotes name of key or character when char cannot be represented in document. Unlike bindings, it will not be shown in bold.

---

1 Fundamentals

1.1 Shells

Command	Description
tclsh *options ?fileName? ?arg ...?*	Tclsh is the non-graphical shell used to evaluate *fileName*. Without *fileName*, it runs interactively, reading Tcl commands from stdin and printing command results and error messages to stdout. For interactive sessions, .tclshrc (or tclshrc.tcl on the Windows) in the home directory of the user is sourced before evaluating *fileName*. Valid options are:
-encoding name	(Tcl 8.5+) Encoding of fileName.
wish *options ?fileName? ?--? ?arg ...?*	Wish is the Tk graphical shell for Tcl, which creates a widow at startup then evaluates *fileName*. Without *fileName* or if the first arg is "--", it runs interactively, reading Tcl commands from stdin and printing command results and error messages to stdout. For interactive sessions, .wishrc (or wishrc.tcl on the Windows) in the home directory of the user is sourced before evaluating *fileName*.
-colormap *new*	Use *new* private colormap instead of using the default colormap for the screen.
-display *display*	*Display* and screen on which to display window
-encoding name	(Tcl 8.5+) Encoding of fileName.
-geometry *geometry*	Initial *geometry* to use for window
-help	Show list of valid options
-name *name*	Use *name* as the title to be displayed in the window, and as the name of the interpreter for send commands.
-sync	Execute all X server commands synchronously and report errors immediately.
-use *id*	Specifies that the window *id* to embed the application main window, instead of creating a independent toplevel window. *Id* must be specified in the same way as the value for the **-use** option for toplevel widgets (i.e. it has a form like that returned by the **winfo id** command).
-visual *visual*	Specifies the visual to use for the window. See Screen or Window Visuals in Toplevel for *visual* options.

Shell Provided Variables

Variable	Description
argc	Number of command line arguments not including the name of the script file.
argv	List of command line arguments.
argv0	Name of script the interpreter is executing or command interpreter if interactive.
geometry	Value of **-geometry** option. (wish only)
tcl_interactive	Returns 1 if the shell is interactive, otherwise 0.

1.2 System Variables

All TCL/TK variables exist in the global namespace unless otherwise specified.

TCL Variables

Variable	Description	
auto_execs	(8.4+) Array of cmd locations as defined by **auto_execok**.	
auto_index	Array of procedures taken from package require commands for **auto_load**.	
auto_noexec	If set, **unknown** will not auto exec external programs.	
auto_noload	If set, **unknown** will not auto load procedures.	
auto_path	List of directories in which package looks for pkgIndex.tcl files when loading packages. Default paths are: **$env**(**TCLLIBPATH**),**$TCL_LIBRARY**, **$TCL_LIBRARY**/.., and **$tcl_pkgPath**. Search will also include all immediate subdirectories. Application specific directories can be appended if necessary.	
env(*var*)	Array where each element name is an enviroment variable. Typical env vars:	
	HOME	User's home directory
	HOSTNAME	Name of machine
	TZ	Time Zone. See clock command for valid time zones.
env(TCL_LIBRARY)	If set, specifies the location of the directory containing library scripts.	
env(TCLLIBPATH)	If set, it must contain a valid Tcl list giving directories in Tcl format with "/" path separators to search during auto-load operations. Used to initialize the **auto_path** variable.	
errorCode	Set to contain a list of one or more elements based on the last Tcl error. Possible values are:	
	ARITH *code msg*	Arithmetic error where *code* is **DIVZERO** (attempt to divide by zero), **DOMAIN** (arg is outside the domain of a function, such as acos(-3)), **IOVERFLOW** (integer overflow), **OVERFLOW** (floating-point overflow), or **UNKNOWN** (cause of the error cannot be determined)
	CHILDKILLED *pid sigName msg*	Child process killed because of a signal.
	CHILDSTATUS *pid exitCode*	Child process has exited with a non-zero exit status.
	CHILDSUSP *pid sigName msg*	Child process has been suspended because of a signal.
	NONE	No additional information is available.
	POSIX *errName msg*	Error occurred during a POSIX kernel call.
errorInfo	Set to the lines of nested code (stack trace) that were being executed when the most recent error occurred.	

TCL_LIBRARY	Location of standard Tcl libraries used for auto loading procedures. Set to first dir the Tcl startup script is found in from **$env**(**TCL_LIBRARY**), compiled in default, location of app, or current dir.	
tcl_nonwordchars	(8.4+) Set to regular expression for control what are considered "nonword" characters (default is anything but Unicode word character or Unicode space on Windows). Auto loaded by use of tcl_endOfWord, etc.	
tcl_patchLevel	Current patch level of Tcl interpreter.	
tcl_pkgPath	List of directories to search for package loading. Typically it contains two directory entries for the location of the platform-dependent and platform independent packages.	
tcl_platform	Array with elements:	
	byteOrder	Set to: **littleEndian** or **bigEndian**
	debug	(8.0.4, 8.0.5, 8.2+) Exists and is set to true, only if debug is enabled
	isWrapped	Set to **wrapped Tcl appd** if wrapped.
	machine	**68k**, **alpha**, **intel**,**mips**, **ppc**, **sparc**, or the result of 'uname -m' on UNIX
	os	Set to: **Windows 95**, **Windows NT**, **MacOS**, **Darwin**, **SunOS**, **Linux**, or the result of 'uname -s' on UNIX.
	osVersion	Set to version or the result of 'uname -r' on UNIX
	platform	Set to: **unix**, **macintosh**, or **windows**
	threaded	(8.2+) Exists and is set to true, only if threads are enabled
	user	(8.1+) Set to user id
	wordSize	(8.4+) Set to size of word in bytes
tcl_precision	Number of significant digits to retain when converting floating-point numbers to strings (default is 12 and IEEE double uses 17). In TCL 8.0p2 this is harded coded to 12.	
tcl_prompt1	Script to output a prompt. Tcl will call script instead of outputting normal prompt.	
tcl_prompt2	Used in a similar way to **tcl_prompt1** when a newline is typed but the current command isn't yet complete. If **tcl_prompt2** isn't set then no prompt is output for incomplete commands.	
tcl_rcFileName	(8.4+) Startup Resouce filename.	
tcl_rcRsrcName	(8.4+) Mac startup resource filename.	
tcl_traceCompile	Level of tracing info (default is 0 or none) output during bytecode compilation. 1 is 1 line per command, and 2 is detailed listing of bytecodes.	
tcl_traceExec	Level of tracing info (default is 0 or none) output during bytecode execution. 1 is 1 line per procedure call, 2 is 1 line per command, and 3 is detailed listing (per instruction).	
tcl_version	Current version of Tcl interpreter in major.minor form.	
tcl_wordchars	(8.4+) Set to regular expression for control what are considered "word" characters (default is Unicode word character or anything but Unicode space on Windows). Auto loaded by use of tcl_endOfWord, etc.	
unknown_pending	(8.4+) Used by unknown to record the command(s) for which it is searching	

TK Variables

Variable	Description
env(TK_LIBRARY)	If set, specifies the location of the directory containing library scripts.
tk_library	Location of standard Tk libraries used for auto loading procedures. Set to first dir the Tk startup script is found in from **$env**(**TK_LIBRARY**), compiled in default, location of Tcl library, location of app, or current dir.
tk_patchLevel	Current patch level of Tk interpreter.
tkPriv	(up to 8.3.5) Array containing information private to standard Tk scripts.
tk::Priv	(8.4+) Array containing information private to standard Tk scripts.
tk_strictMotif	When non-zero, Tk tries to adhere to the Motif look-and-feel as closely as possible.
tk_textRedraw	(8.4+) Set by text widgets when they have debugging turned on.
tk_textRelayout	(8.4+) Set by text widgets when they have debugging turned on.
tk_version	Current version of Tk interpreter in major.minor form.

1.3 Syntax

The following rules define the syntax and semantics of the Tcl language. There may be any number of variable substitutions within a single word but each character is processed only once by the Tcl interpreter as part of creating the words of a command. Substitutions will not affect the word boundaries of a command except for argument expansion. Any well-formed list is also a well-formed command; where if evaluated, each element of the list will become exactly one word of the command with no further substitutions. A Tcl script consists of one or more commands or comments.

Syntax	Description
; or <newline>	Command statement separator except within quotes or braces.
\<newline>	Command statement continuation when at end of line
<white-space>	Command word separator (spaces and tabs only)
#	Comments out rest of line if first non white-space character. The interpreter will still eval braces if present.
"*$var*"	Quoting with substitutions (command, backslash, and variable). Contents of quotes are considered one word and substitutions will be performed by the interpreter. Requires a space between groupings of quotes.
{expand}<non-whitespace>	(Tcl 8.5+) Argument expansion. Removes {expand} then parses and substitutes the rest of the word as any other other command word. After substitution, the word is parsed again without substitutions, and its words are added to the command being substituted.
{*$var***}**	Quoting with deferred substitutions except for newline substitution and {expand}<non-whitespace>. Contents of braces are considered one word and substitutions will be deferred by the interpreter so they can be evaluated later. Used to produce empty string with {}. Can be nested. Requires a space between groupings of braces.
[expr *2+3***]**	Command substitution. Evaluate the command and substitute the result. Interpreter does not perform backslash or variable substitutions before evaluating the command or on the results. Substitutions will occur during the command evaluation. Can be nested.
var	Simple variable. Variable name can consist of letters, digits, underscores, but cannot start with a digit. Can include namespace qualifiers "::".
var(*index*)	Associative array variable where index is element of array var. Same naming standards as *var*.
var(*a,b*)	Pseudo multi-dimensional array variable. Same naming standards as *var*.
$*var*, **$***var***(***index***)** **${***var***},** **${***var***}**($indexVar)	Variable substitution. Replaces variable name with contents of variable without further evaulation by the interpreter. Can include namespace qualifiers "::". Variable names are case sensitive.
\<*char* >	Backslash substitution of <*char*>. Prevents interpretation of special characters. **\a** alert or bell (0x07) \<space> space **\b** backspace (0x08) \\ backslash **\f** form feed (0x0c) *ooo* 8-bit octal value (o=0-7). 1 to 3 digits. **\n** newline (0x0a) **x**hh* 8-bit hexadecimal value (h=0-9, a-f). 1 to 2 digits. **\r** carriage return (0x0d) **u**hhhh* 16-bit unicode hexadecimal value (h=0-9, a-f) (TCL 8.1+) **\t** horizontal tab (0x09) \<*char*> Prevents special meaning of $, ", {, }, [,], etc. **\v** vertical tab (0x0b)

1.4 Operators and Expressions

Operands

The only data type in Tcl is a string. However, Tcl 8.0+ will also keep a native unit representation of a parameter for faster processing if the parameter is not used as a string. Some commands will interpret arguments as numbers/boolean in which case the formats are:

Type	Description
Integer	123 (dec with no preceeding zero), 0xff (hex), 0377 (octal has preceeding zero)
Floating Point	2.1, 3., 4.5e6, 7.8e+9
Boolean	False = **0**, **false**, **no**, **off**; True = **true**, **1**, **yes**, **on** (All versions of expr, only Tcl 8.4+ supports non-values for the Tcl parser)

Operators

The expr command recognizes the following operators, in decreasing order of precedence. Possible operands are numeric values, Tcl variables (with $), strings in double quotes or braces, Tcl comands in brackets, and mathematical functions.

Operators	Description	Validity
- + ~ !	unary minus, unary plus (Tcl 8.4+), bitwise NOT, logical NOT	int, fp (except ~)
**	(Tcl 8.5+) exponentiation	int, fp
* / %	multiply, divide, remainder	int, fp (except %)
+ -	add, subtract	int, fp
<< >>	bitwise shift left, bitwise shift right	int
< > <= >=	boolean comparisons	int, fp, boolean, string
== !=	boolean equal, not equal	int, fp, boolean, string
eq ne	(Tcl 8.4+) boolean string equal, string not equal	string
in ni	(Tcl 8.5+) List and negated list containment. (string in list)	string, list
&	bitwise AND (both bits)	int
^	bitwise exclusive OR (XOR) (either, but not both bits)	int
|	bitwise inclusive OR (either bit)	int
&&	logical AND (lazy evaluation)	int, fp, boolean
||	logical OR (lazy evaluation)	int, fp, boolean
x ? y : z	if x != 0, then y, else z (lazy evaluation)	int, fp

Math Functions

Math functions wil return an error if the result would cause an overflow.

Fn	Description	Fn	Description
abs(*x*)	Absolute value	**int**(*x*)	Integer portion of float
acos(*x*)	Arc cosine (-1<=x<=1)	**log**(*x*)	Natural logarithm (x>0)
asin(*x*)	Arc sin (-1<=x<=1)	**log10**(*x*)	Base 10 logarithm (x>0)
atan(*x*)	Arc tangent	**pow(*x*,*y*)**	Power (x^y)
atan2(y,*x*)	Rectangular (x,y) to polar (r,th), where th=atan2(y,x)	**rand**( )	Random number from 0 to 1
ceil(*x*)	Next integer > x	**round**(*x*)	Round to nearest integer
cos(*x*)	Cosine	**sin**(*x*)	Sine
cosh(*x*)	Hyperbolic cosine	**sinh(*x*)**	Hyperbolic sine
double(*x*)	Convert x to floating point	**sqrt**(*x*)	Square root (x>=0)
exp(*x*)	Exponential function	**srand**(*x*)	Reset rand seed (x is int)
floor(*x*)	Next integer < x	**tan**(*x*)	Tangent
fmod(*x*,*y*)	Floating point remainder (x/y)	**tanh**(*x*)	Hyperbolic tangent
hypot(*x*,*y*)	Hypotenuse of a right-angled triangle sqrt(x*x+y*y)	**wide**(*x*)	(Tcl 8.4+) Convert to 64-bits wide

Constant	Formula	Constant	Formula
e	exp(1)	Pi	acos(-1)

1.5 Pattern Globbing

Pattern	Description	Applicability
?	match any single character	All
*	match zero or more characters	All
[abc]	match set of characters	All
[a-z]	match range of characters	All
\x	match character *x* used for *?[]\ (Tcl 8.1+ understands the special meaning of \a, \b, \f, \n, \r, \t, \v, etc.)	All
{a,b-z}	match any of strings *a*, *b to z* , etc.	glob only
~/	home directory from **$env(HOME)**	glob only
~user	match *user* ’s home directory	glob only

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

1.6 Regular Expressions

Regular expressions (‘‘RE’’s), as defined by POSIX, come in two flavors: extended REs (‘‘EREs’’) and basic REs (‘‘BREs’’). EREs are roughly those of the traditional egrep, while BREs are roughly those of the traditional ed.

Pattern	Description
regex|regex	match either expression
regex*	match zero or more of regex
regex+	match one or more of regex
regex?	match zero or one of regex
.	any single character except newline
^	match beginning of string
$	match end of string
\c	match character c even if special such as . * ? + [ ] ( ) ^ $ | \ (Tcl 8.1+ understands the special meaning of \a, \b, \f, \n, \r, \t, \v, etc.)
[abc]	match set of characters such as [][{}]
[^abc]	match characters not in set
[a-z]	match range of characters
[^a-z]	match characters not in range
( )	group expressions

1.7 Advanced Regular Expressions

Valid in TCL 8.1.1+. Advanced REs ("AREs"') are basically EREs (extended REs) with some significant extensions. An 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. A quantified atom is an atom possibly followed by a single quantifier. Without a quantifier, it matches a match for the atom.

Quantifiers

Quantifiers restrict the atom match to a subset of possible matches. The nominal qualifiers perfer the largest number of matches and the non-greedy qualifiers perfer the smallest match. The forms using **{** and **}** are known as bounds. The numbers *m* and *n* are unsigned decimal integers with permissible values from 0 to 255 inclusive.

Quantifier	Non-Greedy Quantifier	What Quantified Atom Matches
*****	***?**	a sequence of 0 or more matches of the atom
+	**+?**	a sequence of 1 or more matches of the atom
?	**??**	a sequence of 0 or 1 matches of the atom
{*m***}**	**{***m***}?**	a sequence of exactly *m* matches of the atom
{*m***,}**	**{***m* **,}?**	a sequence of *m* or more matches of the atom
{*m***,***n***}**	**{***m***,***n***}?**	a sequence of *m* through *n* (inclusive) matches of the atom; *m* may not exceed *n*

Atoms

Atom	Description	Greedy Preference
(*re***)**	(where *re* is any regular expression) matches a match for *re*, with the match noted for possible reporting	same as RE
(?:*re***)**	as previous, but does no reporting (a ‘‘non-capturing’’ set of parentheses)	same as RE
( )	matches an empty string, noted for possible reporting	same as RE
(?:)	matches an empty string, without reporting	same as RE
[*chars***]**	a bracket expression, matching any one of the chars (see BRACKET EXPRESSIONS for more detail)	none
.	matches any single character	none
k	(where *k* is a non-alphanumeric character) matches that character taken as an ordinary character, e.g. \\ matches a backslash character	none
c	where *c* is alphanumeric (possibly followed by other characters), an escape (AREs only), see ESCAPES below	none
{	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 (see above)	none
x	where x is a single character with no other significance, matches that character.	none

Simple Constraints

A constraint matches an empty string when specific conditions are met. A constraint may not be followed by a quantifier. The lookahead constraints may not contain back references, and all parentheses within them are considered non-capturing. An RE may not end with ‘\’.

Constraint	Description	Greedy Preference
^	matches at the beginning of a line	none
$	matches at the end of a line	none
(?=*re***)**	positive lookahead (AREs only), matches at any point where a substring matching re begins	none
(?!*re***)**	negative lookahead (AREs only), matches at any point where no substring matching re begins	none

Bracket Expressions

Expression	Description
[*abc***]**	match set of characters such as [][{}-]
[^*abc***]**	match characters not in set such as [^][{}-]
[*a-z***]**	match range of characters. A character class may not be used as an endpoint of a range.
[^*a-z***]**	match characters not in range
[.*ch***.]**	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) (Note: Tcl currently has no multi-character collating elements.)
[[.*ch***.]]**	a collating element within a set
[=*e***=]**	equivalence class, standing for the sequences of characters of all collating elements equivalent to that one, including itself. (Note: Tcl currently implements only the Unicode locale. It doesn't define any equivalence classes.)
[=*e***=]**	equivalence class within a set.
[:*class***:]**	Any character in *class*. See Character Classes below.
[[:*class***:]]**	A character *class* within a set.
[[:<:]]	constraint matching empty strings at the beginning of word (word is **[[:alnum:]_]**)
[[:>:]]	constraint matching empty strings at the end of a word (word is **[[:alnum:]_]**)

Character Classes

Character classes are used to define a set of characters in a cross platform way. Tcl only supports Unicode classes.

Class	Description	Class	Description
alnum	Unicode alphabet or digit characters **[[:alpha:][:digit:]]**	**integer**	Valid Tcl form of integer (**string is** only)
alpha	Unicode alphabet characters **[[:lower:][:upper:]]**	**lower**	Unicode lower-case alphabet characters
ascii	Characters **[\u0000-\u007f]** (7-bit ASCII) (machine specific)	**print**	Unicode printing characters, including space
blank	Space or tab characters (not used by **string is**)	**punct**	Unicode punctuation characters (non-alnum or space) (**string is** only)
boolean	true or false, 0 or 1, yes or no, on or off (**string is** only)	**space**	Unicode white-space characters **[\f\n\r\t\v]**
control	Unicode control characters	**true**	true, 1, yes, on (**string is** only)
digit	Unicode digit charactes (not limited to [0-9])	**upper**	Unicode upper-case alphabet characters
double	Valid Tcl form of double (**string is** only)	**wideinteger**	Valid Tcl wide integer. (string is only)
false	false, 0, no, off (**string is** only)	**wordchar**	Unicode word characters, **[[:alnum:][:punct:]]** (**string is** only)
graph	Unicode printing characters, except space	**xdigit**	hexadecimal digit characters **[[0-9][A-F][a-f]]**

Character-Entry Escapes

Character-entry escapes (AREs only) exist to make it easier to specify non-printing and otherwise inconvenient characters in REs.

Char	Description	Char	Description
\a	alert or bell (0x07)	**\t**	horizontal tab (0x09)
\b	backspace (0x08)	**\u***hhhh*	4 digit (16-bit) hex unicode char (h=0-9, a-f, A-F)
\B	synonym for \ to help reduce backslash doubling in some apps with multiple levels of backslash processing	**\U***hhhhhhhh*	8 digit (32-bit) hex unicode char (h=0-9, a-f, A-F)
\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	**\v**	vertical tab (0x0B)
\e	the character whose collating-sequence name is 'ESC', or failing that, the character with octal value 033	**\x***hh*	? digit hexadecimal char (h=0-9, a-f, A-F)
\f	form feed (0x0C)	**\0**	the character whose value is 0
\n	newline (0x0A)	*oo*	2 digit (6-bit) octal value (o=0-7)
\r	carriage return (0x0D)	*ooo*	3 digit (8-bit) octal value (o=0-7)

Class-Shorthand Escapes

Class-shorthand escapes (AREs only) provide shorthands for certain commonly-used character classes. Within bracket expressions, '\d', '\s', and '\w' lose their outer brackets, and '\D', '\S', and '\W' are illegal.

Char	Description	Char	Description
\d	**[[:digit:]]**	**\D**	**[^[:digit:]]**
\s	**[[:space:]]**	**\S**	**[^[:space:]]**
\w	**[[:alnum:]_]** (note underscore)	**\W**	**[^[:alnum:]_]** (note underscore)

Constraint Escapes

A constraint escape (AREs only) is a constraint, matching the empty string if specific conditions are met. A word is defined as in the specification of **[[:<:]] [[:>:]]**. Constraint escapes are illegal within bracket expressions. A back reference (AREs only) matches the same string matched by the parenthesized subexpression specified by the number. The subexpression must entirely precede the back reference in the RE. Subexpressions are numbered in the order of their leading parentheses. Non-capturing parentheses do not define subexpressions.

Char	Description	Char	Description
\A	matches only at the beginning of the string whereas ^ also matches empty string after a newline	**\Y**	matches only at a point that is not the beginning or end of a word
\m	matches only at the beginning of a word	**\Z**	matches only at the end of the string whereas $ also matches empty string before a newline
\M	matches only at the end of a word	*m*	(where *m* is a nonzero digit) a back reference
\y	matches only at the beginning or end of a word	*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

Metasyntax

Normally the flavor of RE being used is specified by application-dependent means. However, this can be overridden by a director. An ARE may begin with embedded options: a sequence (?xyz) (where xyz is one or more alphabetic characters) specifies options affecting the rest of the RE. These can supplement and/or override any options specified by the application. Embedded options take effect at the ")" terminating the sequence. They are available only at the start of an

ARE, and may not be used later within it.

Director	Description
***	At the start of a RE, then the rest of the RE is an ARE.
***=	At the start of a RE, then the rest of the RE is to be taken to be a literal string, with all characters considered ordinary characters.
b	rest of RE is a BRE
c	case-sensitive matching (usual default)
e	rest of RE is an ERE
i	case-insensitive matching (x becomes [xX] and [^x] becomes [^xX])
m	historical synonym for n
n	newline-sensitive matching ("." and bracket expressions using ^ will never match the newline character. $ and ^ will match the empty string before and after a newline in addition to at the end and beginning of a string respectively)
p	partial newline-sensitive matching ("." and bracket expressions using ^ will never match the newline character.)
q	rest of RE is a literal ("quoted") string, all ordinary characters
s	non-newline-sensitive matching (usual default)
t	tight syntax (usual default; all characters are significant)
w	inverse partial newline-sensitive ("weird") matching ($ and ^ will match the empty string before and after a newline in addition to at the end and beginning of a string respectively)
x	expanded syntax (see below)

Expanded Syntax

When selected by the -expanded switch or x option, white-space (blank, tab, newline, and [[:space:]]) and all characters between a # and the following newline or end of RE are ignored. Exceptions are: when preceeded by a \, within a bracket expression, and within multi-character symbols (illegal).

Comments

Outside bracket expressions, the sequence "(?#ttt)" (where ttt is any text not containing a ")") is a comment and will be ignored. This syntax is deprecated in favor of the expanded syntax.

Matching

In the event that an RE could match more than one substring of a given string, the RE matches the one starting earliest in the string. If the RE could match more than one substring starting at that point, its choice is determined by its preference: either the longest substring, or the shortest. A branch has the same preference as the first quantified atom in it which has a preference. An RE consisting of two or more branches connected by the | operator prefers longest match. Subject to the constraints imposed by the rules for matching the whole RE, subexpressions also match the longest or shortest possible substrings, based on their preferences, with subexpressions starting earlier in the RE taking priority over ones starting later. Note that outer subexpressions thus take priority over their component subexpressions. Match lengths are measured in characters, not collating elements. An empty string is considered longer than no match at all.

---

2 Tcl Commands

2.01 Arrays

Tcl arrays are associative arrays based on a hash table data structure. Elements of an array can consist of any string or number unlike traditional array elements which are integers only. For the array commands below, *arrayName* is the name of the array not the array contents (don't use variable substitution).

Command	Description
array anymore *arrayName searchId*	Returns 1 if more elements are left to be processed in *searchId* of *arrayName* , 0 if none.
array donesearch *arrayName searchId*	Terminates the array search *searchId* on *arrayName*.
array exists *arrayName*	Returns 1 if *arrayName* is an array variable, 0 if not.
array get *arrayName ?pattern?*	Returns a list of all element and value pairs in *arrayName* or those matching *pattern* using Pattern Globbing. The first is the element name and the second is the element value. If no match then an empty string is returned.
array names *arrayName ?mode? ?pattern?*	Returns a list of all element names in *arrayName* or those matching *pattern*. In Tcl 8.4+, *mode* can be **-exact** (same string), **-glob** (default, using Pattern Globbing), or **-regexp** (using Regular Expressions). If no match then an empty string is returned.
array nextelement *arrayName searchId*	Returns name of next element in*arrayName* for the search *searchId* . Returns an empty string if no more elements exist.
array set *arrayName list*	Sets values of one or more elements in *arrayName* from *list* in **array get** format.
array size *arrayName*	Return number of elements in *arrayName*. If not an array then 0 is returned.
array startsearch *arrayName*	Initiates an element-by-element search of *arrayName*. Returns a search id. Muliple searches of same array are supported.
array statistics *arrayName*	(Tcl 8.4+) Returns number of entries in the table, the number of buckets, and the utilization of the buckets of the hash table that represents *arrayName* .
array unset *arrayName ?pattern?*	(Tcl 8.3+) Unsets all of the elements in *arrayName* or matching *pattern* using Pattern Globbing . If *arrayName* is not an array or no match is found, an error is returned.
parray *arrayName ?pattern?*	Print to standard output the names and values of all elements in *arrayName* or matching *pattern* using Pattern Globbing.

2.02 Clock

Tcl does not include any leap seconds in clock values, seconds are counted as if each UTC day has exactly 86400 seconds. Tcl responds to leap seconds by speeding or slowing its clock by a tiny fraction for some minutes until it is back in sync with UTC; its data model does not represent minutes that have 59 or 61 seconds.

UNIX and Windows NT Epoch is 1 January 1970, 00:00 UTC**.** This is the epoch for all systems in Tcl 8.5+.
Julian Epoch is 1 January, 4713 BCE of the proleptic Julian calendar

Command	Description
clock add clockVal ?count unit? ?count unit...? ?-option value? ?-option value...?	(Tcl 8.5+) Add all count unit (can be negative) conversions to integer *clockVal*in the specified order. Count is an integer of type unit. Unit is seconds, minutes, hours, days, weeks, months, or years, or any of their unique prefixes. While leap days and Daylight Savings Time are accounted for in the conversions, leap seconds are not. For ambiguious times where the same local time occurs twice on the same day, the earlier time is used. For impossible times (skipped hour for Daylight Savings Time, etc.), the time is converted as if the clock had not changed.

-gmt *boolean*	If true, use GMT/UTC time zone, if false (default) use local time zone.
-locale name	Specifies that conversions should be done according to the rules of locale name. Valid names are: any valid locale supported by msgcat, "system" to use the current system locale (from LC_TIME env var or Control Panel date/time on MS Windows), or {} to use Tcl's default locale (default for no -locale).
-timezone timeZone	Specifies that conversions should be done according to the rules of Time Zone timeZone. See Time Zones below for the valid time zones. The time zone preference order is: -timezone or -gmt options, TCL_TZ env var, TZ env var, Control Panel time zone on MS Windows, or the C language local time as defined by the localtime and mktime functions.
clock clicks ?-option?	Returns hi-res system-dependent integer time value. In Tcl 8.5+, returned value is a wide int. Options are:
-microseconds	(Tcl 8.5+) Return current time as system-dependent integer value of microsecondssince "epoch".
-milliseconds	(Tcl 8.3+) Return current time as system-dependent integer value of milliseconds since "epoch".
clock format *clockVal* ?-option value? ?-option value ...?	Convert integer *clockVal* in seconds to human-readable format defined by the format string. Valid options are:
-format *string*	Specifies the output format. See Clock Formats below for valid format fields. The default format is "%a %b %d %H:%M:%S %Z %Y" prior to Tcl 8.5 and "%a %b %d %H:%M:%S %z %Y" for Tcl 8.5+.
-gmt *boolean*	If true, use GMT/UTC time zone, if false (default) use local time zone.
-locale name	(Tcl 8.5+) Specifies that conversions should be done according to the rules of locale name. Valid names are: any valid locale supported by msgcat, "system" to use the current system locale (from LC_TIME env var or Control Panel date/time on MS Windows), or {} to use Tcl's default locale (default for no -locale). The current locale can be used with -locale current.
-timezone timeZone	(Tcl 8.5+) Specifies that conversions should be done according to the rules of Time Zone timeZone. See Time Zones below for the valid time zones. The time zone preference order is: -timezone or -gmt options, TCL_TZ env var, TZ env var, Control Panel time zone on MS Windows, or the C language local time as defined by the localtime and mktime functions.
clock microseconds	(Tcl 8.5+) Return current time as system-dependent integer value of microsecondssince "epoch".
clock milliseconds	(Tcl 8.5+) Return current time as system-dependent integer value of millisecondssince "epoch".
clock scan *"dateString"* ?-option value? ?-option value ...?	(Tcl 8.5+) Convert *dateString* to an integer clock value. In Tcl 8.5+, returned value is a wide int. While leap days and Daylight Savings Time are accounted for in the clock add conversions, leap seconds will not.
-base *clockVal*	Use integer *clockVal* (in seconds) as the base for date-relative conversions in dateString.
-format *string*	(Tcl 8.5+)Specifies the input format. See Clock Formats below for valid format fields.
-gmt *boolean*	If true, use GMT/UTC time zone, if false (default) use local time zone.
-locale name	(Tcl 8.5+) Specifies that conversions should be done according to the rules of locale name. Valid names are: any valid locale supported by msgcat, "system" to use the current system locale (from LC_TIME env var or Control Panel date/time on MS Windows), or {} to use Tcl's default locale (default for no -locale).
-timezone timeZone	(Tcl 8.5+)Specifies that conversions should be done according to the rules of Time Zone timeZone. See Time Zones below for the valid time zones. The time zone preference order is: %z or %Z formats, -timezone or -gmt options, TCL_TZ env var, TZ env var, Control Panel time zone on MS Windows, or the C language local time as defined by the localtime and mktime functions.

clock scan *"dateString"* ?**-base** *clockVal*? ?**-gmt** *boolean*?	Convert *dateString* to an integer clock value. If only a time is specified, current date is assumed. Without time, midnight is assumed. Without time zone uses local zone unless **-gmt** is specified. If **-base** is used, the date in *clockVal* is used for determining the time on a specific day or other date-relative conversions (like daylight savings time for day or greater units). Allowed range of years is 1902 to 2037. *DateString* consists of zero or more specifications of the following forms:
time	Time of day form: *"hh?:mm?:ss?? ?meridian? ?zone?"* or "*hhmm ?meridian?" ?zone?* . Without meridian, *hh* is interpreted on a 24-hour clock.
date	Month, day, year forms: "*mm/dd?/yy?", "monthname dd?, yy?", "dd monthname ?yy?", "day, dd monthname yy", "?CC?yymmdd"* (Tcl 8.3+), *"?CC?yy-mm-dd"* (Tcl 8.3+), *"dd-monthname-?CC?yy"* (Tcl 8.3+). Default *yy* is current year. If *yy*< 100, 00-38 is 2000-2038 (prior to Tcl 8.3), 00-68 is 2000-2068 (Tcl 8.3+), 69-99 is 1969-1999.
ISO-8601-point-in-time	(Tcl 8.3+) ISO 8601 format: *"CCyymmdd***T***hhmmss", "CCyymmdd hhmmss", or "CCyymmdd* **T***hh:mm:ss*".
relative time	Relative to current time. Format is *number unit*. Units are: **year**,**fortnight**, **month**, **week**, **day**, **hour**, **minute** (or **min**), and **second** (or **sec**) and their plurals, with modifiers: **tomorrow**,**yesterday**, **today**, **now**, **last**, **this**, **next**, and **ago**. Daylight savings time correction is applied only for day, week, fortnight, month, or year.
stardate *float*	(Tcl 8.3+) Returns time in Star Trek stardate floating point format.
now	Use current time
clock seconds	Return current time as system-dependent integer value of seconds since "epoch". In Tcl 8.5+, returned value is a wide int.

Clock Formats

Field	Description	Field	Description	Field	Description
%%	%	%j	Day of Year (001-366)	%t	(All UNIX,845+ MS Win) Tab
%a	Weekday (abbr)	%J	(Tcl 8.5+) Julian Day Number	%T	(All UNIX,8.4+ MS Win) Locale Time. "C" locale default:" %H:%M:%S"
%A	Weekday (full)	%k	(Tcl 8.4+) Hour (0-23)	%u	(Tcl 8.4+) Weekday (1-7), 1=Mon
%b	Month (abbr)	%l	(Tcl 8.4+) Hour (1-12)	%U	Week (00-53), starts on Sun
%B	Month (full)	%m	Month (01-12)	%V	(Tcl 8.3+) Week (00-52), Week 1 contains Jan 4. ISO8601 fiscal week.
%c	Locale date & time. "C" locale default: "%a %b%d %Y %I:%M:%S %p %Z"	%M	Minute (00-59)	%w	Weekday (0-6) 0=Sun
%C	Year prefix (19 or 20)	%n	(All UNIX,8.4+ MS Win) Newline	%W	Week (00-53), starts on Mon
%d	Day (01-31)	%N	(Tcl 8.5+) Month number (1-12)	%x	Locale Date. "C" locale default: "%m/%d/%y"
%D	(All UNIX,8.4+ MS Win) Locale Date. "C" locale default: "%m/%d/%y"	%O#	(Tcl 8.5+) Locale alt numerals for d, e, H, I, k, l, m, M, S, u, w, y	%X	Locale Time. "C" locale default:" %I:%M:%S %p"
%e	(All UNIX,8.4+ MS Win) Day of month (1-31)	%p	Locale AM/PM	%y	Year (00-99)
%E#	(Tcl 8.5+) Locale's alt calendar for c, C, x, X, y, Y	%P	(UNIX only) Locale am/pm	%Y	Year (full)
%g	(Tcl 8.4.7+) Year for %V (00-99)	%Q	(Tcl 8.3+) Stardate	%z	(Tcl 8.5+) Time Zone Offset in +/-hhmm from GMT
%G	(Tcl 8.4.7+) Year for %V (full)	%r	(All UNIX,8.4+ MS Win) Locale meridian time. "C" locale default: "%I:%M:%S %p".	%Z	Time Zone name
%h	(All UNIX,8.4+ MS Win) Month (abbr)	%R	(All UNIX,8.4+ MS Win) Locale Time. "C" locale default: %H:%M	%+	(Tcl 8.5+) Date/Time '%a %b %e %H:%M:%S %Z %Y'
%H	Hour (00-23)	%s	(Tcl 8.3+) Seconds since epoch		
%I	Hour (01-12)	%S	Seconds (00-59)		

where locale defaults are based on the environment variables LC_ALL and LC_TIME.

Time Zones

adt - Atlantic Daylight Time	east - Eastern Australian StandardTime	mdt - Mountain DaylightTime	swt - Swedish WinterTime
ahst - Alaska-Hawaii Standard Time	edt - Eastern DaylightTime	mest - Middle European SummerTime	ut - Universal (Coordinated)
ast - Atlantic Standard Time	eest - Eastern European Summer Time	met - Middle EuropeanTime	utc - Universal Coordinated Time
at - AzoresTime	eet - Eastern EuropeTime, USSR Zone 1	mewt - Middle European WinterTime	wadt - West Australian DaylightTime
bst - British Summer Time	est - Eastern StandardTime	mst - Mountain StandardTime	wast - West Australian StandardTime
bt - BaghdadTime, USSR Zone 2	gmt - Greenwich MeanTime	ndt - Newfoundland Daylight	wat - West Africa Time
cadt - Central Australian Daylight Time	gst - Guam StandardTime, USSR Zone 9	nft - NewfoundlandTime	wet - Western EuropeanTime
cast - Central Australian Standard Time	hdt - Hawaii DaylightTime	nst - Newfoundland StandardTime	ydt - Yukon DaylightTime
cat - Central Alaska Time	hst - Hawaii StandardTime	nt - Nome Time	yst - Yukon StandardTime
cct - China Coast Time, USSR Zone 7	idle - International Date Line East	nzdt - New Zealand DaylightTime	zp4 - USSR Zone 3
cdt - Central Daylight Time	idlw - International Date Line West	nzst - New Zealand StandardTime	zp5 - USSR Zone 4
cest - Central European Summer Time	ist - Indian StandardTime	zt - New Zealand Time	zp6 - USSR Zone 5
cet - Central European Time	it - Iran Time	pdt - Pacific DaylightTime	
cst - Central Standard Time	jst - Japan StandardTime, USSR Zone 8	pst - Pacific StandardTime	
eadt - Eastern Australian Daylight Time	jt - Java Time	sst - Swedish SummerTime	

In Tcl 8.5+, the following forms are supported:

Format	Description	Examples
Name	Time Zone Acronym (see table above)	UTC, CDT
:name	Locale Time Zone. Special case of :localtime (local time per C library). For a complete listing, see: "/no_backup/tools/lib/tcl8.5/clock/tzdata" for Non-UNIX or "/usr/share/zoneinfo" for UNIX.	:UTC, :America/New_York
+/-#### +/-######	Time Zone Offset in hours, minutes, and seconds (if six digits are present) from UTC. Use a plus sign for east of GMT and a minus sign for west of GMT.	+0500, -063000
std offset ?dst offset,rule?	Posix specification of the TZ environment variable	

2.03 Command Evaluation

Command	Description
auto_execok *cmd*	Returns full pathname of *cmd* for use by **exec** if it exists in the dirs specified by **$env**(**PATH**) or is built-in, otherwise returns an empty string. Only finds files with execute bits set.

auto_import *pattern*	(Tcl 8.0.3+) Search **auto_index** array and forcably load procedures matching *pattern*. In Tcl 8.3.4+, uses **namespace import** style matching.
auto_load *cmd*	Attempts to load the definition for *cmd* by searching **$auto_path** then **$env(TCLLIBPATH)** for a **tclIndex** file which defines the location and script to load*cmd*. Returns 1 if successful, 0 if not.
auto_mkindex *dir pattern ?pattern ...?*	(Tcl 8.3+) Generate a tclIndex file from all files in the specified directory matching **glob** *pattern*s for use by auto_load.
auto_mkindex_old *dir args*	(Tcl 8.3+, was **auto_mkindex** prior to 8.3) Generate a tclIndex file from all files in the specified directory. Only procedures with "proc" at the beginning of a line (no leading spaces) are included.
auto_reset	Destroys cached information used by **auto_execok** and **auto_load**.
bgerror *message*	(Undefined for TCL) User defined handler for background Tcl errors. Default for Tk is to post dialog box with error message and ask if stack trace should be shown. The **errorInfo** and **errorCode** variables are set to their values at the time the error occurred before calling **bgerror**.
catch *script ?resultVarName? ?optionsVarName?*	Evaluate *script* and trap any errors. If there is an error, the non-zero error code (see return) is returned and the error message is stored in *resultVarName*. If not, 0 (TCL_OK) is returned with *resultVarName*set to the value returned from the script. Within *script*,**break** can be used to terminate the script. In Tcl 8.5+, *optionsVarName* is set to a dictionary of the return options returned by evaluation of script. If the error code is TCL_RETURN, the options dict will set -code and -level to values set by the return command. For all other errors, -code will be set to the error code and -level will be 0. For TCL_ERROR, the dict will also include -errorinfo (contents of ::errorInfo), -errorcode (contents of ::errorCode), and -errorline (line of script where error occurred).
error *message ?info? ?code?*	Interrupt command interpretation and pass back the error described in *message*. Global variables **errorInfo** and **errorCode** will be set to *info* and *code* if defined.
eval *arg ?arg ...?*	Returns result of evaluating the concatenation of *args* as a Tcl command.
exit *?returnCode?*	Terminate the process, returning*returnCode* (default is 0) to the system as the exit status. UNIX limits range from 0 to 255.
expr *arg ?arg ...?*	Concatenates *arg*s with separators, evaluates the result as a Tcl expression, and returns the value. See Operators and Math Functions for more info. Tcl 8.3.3+ allows for setting variables via command substution within an expression. To do numeric comparisons, all values must be numeric. To return a result in floating point format, at least one value must be in floating point format. The precision is determined by the contents of the **tcl_precision** variable. In TCL 8.0+, it is more efficient to group expressions within braces {} to let **expr** perform substitutions. To compare strings, use quotes around the strings. In TCL 8.4+, "nan" is recognized as Not a Number and "inf" or "infinity" is recognized as infinite.
load *fileName pkgName ?interp?*	Load binary code (shared library) for *pkgName* from *fileName* into *interp* (default is current). If *fileName* is an empty string, Tcl uses *pkgName* to find matching statically linked then dynamic library. Without *pkgName*, Tcl guesses the name.
rename *oldName newName*	Rename command *oldName* to *newName*. If *newName* is the empty string, command *oldName* is deleted. Can include namespace qualifiers.
send *options ?--? interp command ?arg ...?*	(Tk UNIX only) Evaluates *command* with *args* in the Tk application *app* (set with **tk appname** command) on the same display and returns the result or command execution error. *Options* are:
-async	Will complete immediately without waiting for*command* to complete in the target application. No results will be available and errors will be ignored.
-displayof *window*	Use *window*'s display instead of the current display.
source *fileName*	Read file *fileName* and evaluate its contents as a Tcl script. Returns the return value of last comand in script or error if one occurs. For MS Windows and all platforms in Tcl 8.4+, the EOF is set to \x1a.
source -encoding*encodingName fileName*	(Tcl 8.5+) Read file *fileName* in encoding encodingName (defult is system encoding) and evaluate its contents as a Tcl script. Returns the return value of last comand in script or error if one occurs. Default EOF is set to \x1a.

subst ?**-nobackslashes**? ?**-nocommands**? ?**-novariables**? *string*	Returns result of backslash, command, and variable substitutions on *string*. Each substitution type may be turned off by the corresponding option. Except for command, the "{} chars do not have a special meaning.
tcl_findLibrary *basename version patch initScript envVarName varName*	(Tcl 8.0.3+) Used by extensions to look for their script library. Uses *basename* and *version* for directory name. The *initScript* file will be sourced into the interpreter and the directory will be stored in the global variable *varName* unless *varName* is already defined. Checks directories: directory from **env(***envVarName***)**; relative to Tcl library directory; relative to the executable file in the standard installation bin or bin/ *arch* directory; relative to the executable file in the current build tree; relative to the executable file in a parallel build tree.
time *script ?count?*	Call interpreter *count* times (default is 1) to evaluate *script*. Returns string of the form "# microseconds per iteration".
unknown *cmdName ?arg ...?*	Called when the Tcl interpreter encounters an undefined *cmdName*. Default **unknown** calls **auto_load** then **auto_exec** to load or exec *cmdName* with *args*. If not successful and called from top-level but outside of a script, it checks for csh like-history substitution forms of **!!**, **!***event* , or ^*old*^*new*?^?. If found it performs the history substitution. Lastly it checks if *cmdName* is a unique abbreviation of an existing Tcl command and if so expands the command name and executes it. If none were successful, an error is returned.
unload *fileName ?pkgName? ?interp?*	(Tcl 8.5+) Unload package pkgName from shared library filename previously loaded with load from interp interp.Without interp, the current interp is used. Without *pkgName*, Tcl guesses the name in the same manner as load.
-nocomplain	Supresses all error messages.
-keeplibrary	Prevents unload from issuing the operating system call that will unload the library from the process.

2.04 Control Loops

Command	Description
break	Abort innermost loop (for, foreach, while, catch) or tag for a Tk binding script containing command.
case	Obsolete, use **switch**.
continue	Skip to the next iteration of innermost loop (for, foreach, while) or tag for a Tk binding script containing command.
for *{start} {test} {next} {body}*	First evaluate *start* then repeatedly evaluate *body* then *next* if **expr** *test* returns a non-zero result. If strings are used as operands in the expression, they must be quoted or in braces.
foreach *varname {list} {body}*	For each item in *list*, set *varname* to the item's value and evaluate *body*.
foreach *{varlist1} {list1} ?{varlist2} {list2} ...? {body}*	Same as above, except for each iteration of the loop, the variables in *varlistN* are set to the next entry in their corresponding *listN*.
if {expr1} ?**then**? *{body1}* **elseif** *{expr2}* ?**then** ?*{body2}* ... ?**else**? ?{*bodyN*}?	If *expr1* evaluates **true**, *body1* is evaluated, otherwise if *expr2* is **true**, *body2* is evaluated, etc. If none of the expressions evaluate to **true** then *bodyN* is evaluated. If strings are used as operands in the expression, they must be quoted or in braces.
switch *?options? ?--? string pattern1 {body1} ?pattern2 {body2} ...?*	For the first *pattern* that matches *string* , evaluate the corresponding *body* and return result. If no pattern is matched and **default** is the last pattern, then its body is evaluated, otherwise an empty string is returned. If *body* is set to " -", the body for the next *pattern* that isn't "-" will be used. Options are:
-exact	*String* must contain exactly the same string as *pattern*. This is the default option.
-glob	Compare *patterns* to *string* using Pattern Globbing.
-regexp	Compare *patterns* to *string* using Regular Expression pattern matching.
-matchvar varName	(Tcl 8.5+) Used with -regexp, to specify the variable name to store the list of the matches found by the regular expression engine. List args are same as the results stored to *matchVar and subMatchVars* in regexp command. Will be set to empty list for default case.
-indexvar varName	(Tcl 8.5+) Used with -regexp, to specify the variable name to store the list of indices (same form as regexp -indicies) referring to matching substrings found by the regular expression engine (see -matchvar).Will be set to empty list for default case.
switch *?options? ?--? string {pattern1 body1 ?pattern2 body2 ...?}*	Same as above except patterns and bodies are evaluated as a concatenated list of all patterns and commands with no command or variable substitutions performed.
while *{test} {body}*	As long as expression *test* evaluates to **true**, evaluate Tcl command string *body*. If strings are used as operands in the expression, they must be quoted or in braces.

2.05 Dictionary

Dictionaries are values that contain an efficient (but not order-preserving) mapping from arbitrary keys to arbitrary values. They have a textual format that is exactly that of any list with an even number of elements (a.k.a. keyed list), with each mapping in the dictionary being represented as two items in the list. In the commands below, dict is the contents of a dictionary (variable substitution, etc.) and dictName is the name of a dictionary variable.

Command	Description
dict append dictName key ?string ...?	Appends string or strings to key's value in dictionary dictName. Non-existent keys are treated as {}.
dict create ?key value ...?	Returns a new dictionary that contains each of the specified key and value mappings.
dict exists dict key ?key ...?	Returns 1 if dict contains key (or path of keys through a set of nested dictionaries) , or 0 if it does not.
dict filter dict filterType arg ?arg ...?	Returns a new dictionary that only contains the key/value pairs that match filterType in dict. Valid filterTypes are:
key pattern	Include elements where the key matches *pattern* using Pattern Globbing.
script {keyVar valueVar} script	Include elements where the result of evaluating script is 1. Filtering is performed by looping through each dict element and setting keyVar to the key and valueVar to the value then evaluating script.If script returns TCL_BREAK, no further key/value pairs are checked or included. TCL_CONTINUE is equivalent to a false result.
value pattern	Include elements where the value matches *pattern* using Pattern Globbing.
dict for {keyVar valueVar}dict body	Loop through each dict element and set keyVar to the key and valueVar to the value then evaluating body. If body returns TCL_BREAK, no further key/value pairs will be iterated over. TCL_CONTINUE is equivalent to TCL_OK.
dict get dict ?key ...?	Returns the value for key (or path of keys through a set of nested dictionaries) in dict. Without key, a list of all key/value pairs in *dict* is returned. Non-existant keys return an error.
dict incr dictName key ?increment?	Increments the value of key by value (defaults to 1) in dictionary dictName.Non-existent keys are treated as if they map to 0. An error is returned if key's value is not an integer.
dict info dict	Returns implementation specific info about dict.
dict keys dict ?pattern?	Returns a list of all keys in dict matching *pattern* using Pattern Globbing. The keys are in an arbitrary order. Without pattern, all keys are returned in the same arbitrary order as dict values.
dict lappend dictName key ?value ...?	Appends each value to key's list value in dictionary dictName.Non-existent keys are treated as if they map to an empty list. An error is returned if key's value can not be represented as a list.
dict merge ?dict ...?	Returns a dictionary containing the contents of all dict's. For duplicate keys, only the value from the last dictionary with key is used.
dict remove dict ?key ...?	Returns a dictionary without keys.
dict replace dict ?key value ...?	Returns a dictionary that adds to or replaces each key and value pair in dict.
dict set dictName key ?key ...? value	Sets (add or replace) the key (or path of keys through a set of nested dictionaries) in dictionary dictName with value.
dict size dict	Returns the number of key/value mappings in dict.
dict unset dictName key ?key ...?	Unsets (removes) the key (or path of keys through a set of nested dictionaries) in dictionary dictName. At least one key must be specified, but the last key on the key-path need not exist.
dict update dictName key varName ?key varName ...? body	Map each varName to key then evaluate and return the result of body. If a key does not exist, then varName is unset.When done evaluating body, any changes made to the varNames are reflected in dictionary dictName.
dict values dict ?pattern?	Returns a list of all values in dict matching *pattern* using Pattern Globbing. The values are in an arbitrary order. Without pattern, all values are returned in the same arbitrary order as dict keys.
dict with dictName ?key ...? body	Map each key in dictionary dictName (or chain of nested dictionaries if one or more keys are used) to a variable with the same name then evaluate and return the result of body. When done evaluating body, any changes made to the variables are reflected in dictionary dictName.

2.06 Encodings

Command	Description
encoding convertfrom *?encoding? data*	(Tcl 8.1+) Convert data to Unicode from the specified encoding. Uses current system encoding if not specified.
encoding convertto *?encoding? string*	(Tcl 8.1+) Convert string from Unicode to the specified encoding. Uses current system encoding if not specified.
encoding names	(Tcl 8.1+) Return list of all available encodings.
encoding system *?encoding?*	(Tcl 8.1+) Set the system encoding to encoding. Returns current encoding if *encoding* is not specified.

Common Encodings

Type:	Example Encoding Names:				
Single Byte:	ascii	cp1252(MS Windows)	iso8859-1	symbol	utf-8
Double Byte:	unicode	big5 (chinese)			
Variable Byte:	shiftjis	euc-jp			
3 or more bytes:	Invalid				

2.07 Event Loop Handlers

Command	Description
after *ms*	Sleep for *ms* milliseconds. Blocks during sleep.
after *ms?arg1 arg2 ...?*	Arrange for command (concat of *arg*s) to be run after *ms* milliseconds have passed as an event handler. Returns the ID of the event handler created. Does not block.
after cancel *ID*	Cancel previous **after** command with *ID*.
after cancel *arg1 arg2 ...*	Cancel previous **after** command matching *arg*s.
after idle *?arg1 arg2 ...?*	Arrange for command (concat of args) to be evaluated later as an idle callback (TK is idle). Returns the ID of the event handler created. Do not call another after idle from an after idle callback. Use after 0 instead.
after info *?ID?*	Returns information on event handler *ID*. With no *ID*, returns a list of all existing event handler IDs. Each list entry contains two elements consisting of the script and event handler type.
tkwait variable *varName*	(Tk only) Wait for global variable *varName* to be modified before proceeding. Does not block while waiting, but nested tkwaits must complete before outer wait can complete.
tkwait visibility *window*	(Tk only) Waits for a change in the visibility state of *window* before proceeding. Can be used to wait for a window to be created before taking action. Does not block while waiting, but nested tkwaits must complete before outer wait can complete.
tkwait window *window*	(Tk only) Waits for *window* to be destroyed before proceeding. Can be used to wait for a dialog to be closed before taking action. Does not block while waiting, but nested tkwaits must complete before outer wait can complete.
update ?**idletasks**?	Handle pending events including idle callbacks. If **idletasks** is specified, only those operations normally deferred (idle callbacks, display updates, and window layout calcs) until the idle state are processed.
vwait *varName*	Enter Tcl event loop until global or fully qualified namespace variable or array *varName* is modified. Will block if no events are ready and nested vwaits must complete before outer wait can complete.

2.08 File Attributes

<table>
<tr><td>Command</td><td>Description</td></tr>
<tr><td>file atime
fileName?time?</td><td>Returns the time that fileName was last accessed as seconds since system epoch time. In Tcl 8.3+, time sets last accessed time. On Windows, FAT file systems do not support access time.</td></tr>
<tr><td>file attributes
fileName ?option? ?option value? ...</td><td>Sets platform-specific attribute option to value for fileName. Without value, returns current value. Without option, returns all options and values. Valid options are:</td></tr>
<tr><td>-archive boolean</td><td>(MS Windows) Archive file</td></tr>
<tr><td>-creator type</td><td>(Mac, Mac OS X (8.5+)) Creator type</td></tr>
<tr><td>-group name</td><td>(UNIX) Group Name. Group ID can be used for set, but only names are returned.</td></tr>
<tr><td>-hidden boolean</td><td>(Mac, Mac OS X (8.5+), MS Windows) Hidden file</td></tr>
<tr><td>-longname filename</td><td>(MS Windows) Filename, cannot be set</td></tr>
<tr><td>-owner name</td><td>(UNIX) Owner name. Owner ID can be used for set, but only names are returned.</td></tr>
<tr><td>-permissions code</td><td>(UNIX) Permissions in octal format. Tcl 8.3+ adds limited support for symbolic attributes like chmod or an ls style string of the form rwxrwxrwx (must be 9 characters). Symbolic attributes syntax is: [ugoa][+-=][rwxst],?...? where the comma separates multiple attributes.
<table>
<tr><td>Field</td><td>File</td><td>Directory</td></tr>
<tr><td>User, Group, Others</td><td>r = view file contents, w = modify file contents, x = execute file</td><td>r = view dir contents, w = modify dir contents, x = view dir contents and access dir’s files</td></tr>
<tr><td>Set UID</td><td>set user to file’s owner at runtime
(s if x, S if no x)</td><td></td></tr>
<tr><td>Set GID</td><td>Set group to file’s group at runtime
(s if x, S if no x)</td><td>All files created in dir will inherit the group of the dir</td></tr>
<tr><td>Sticky</td><td>(obsolete) File should "stick" in memory after it is finished executing
(t if x, T if no x)</td><td>(system dependent) User can create/modify files in dir with write access, but can only delete files they own.</td></tr>
</table></td></tr>
<tr><td>-readonly boolean</td><td>(Mac, Mac OS X (8.5+), BSD UNIX, MS Windows) Read-only or UNIX user immutable flag</td></tr>
<tr><td>-rsrclength length</td><td>(Tcl 8.5+ Mac. Mac OS X) Length of the resource fork of a file, can only be set to 0</td></tr>
<tr><td>-shortname filename</td><td>(MS Windows) Filename, cannot be set</td></tr>
<tr><td>-system boolean</td><td>(MS Windows) System file</td></tr>
<tr><td>-type type</td><td>(Mac, Mac OS X (8.5+)) Finder type</td></tr>
<tr><td>file channels
?pattern?</td><td>(Tcl 8.3+) Returns a list of all open I/O channels (files, sockets, stdio, etc.) or those matching pattern using Pattern Globbing.</td></tr>
<tr><td>file copy ?-force?
?--? source target</td><td>Copies source file or directory to target . Will not overwrite existing files unless -force is specified. In Tcl 8.5+, will copy finder attributes.</td></tr>
<tr><td>file copy ?-force?
?--? source ?source ...? targetDir</td><td>Copies each source file or directory to targetDir directory. If source is a directory, all files in source will be recursively copied to targetDir. Will not overwrite existing files unless -force is specified. Will stop at first error. Invalid operations are: overwrite non-empty directory, overwrite directory with file, or overwrite file with directory.</td></tr>
<tr><td>file delete ?-force ?
?--? fileName ?fileName ...?</td><td>Removes given files or directories. Use -force to remove non-empty directories. For symbolic links, only the link will be deleted. Deleting a non-existent file is not considered an error. Args are processed in the order specified, halting at the first error, if any.</td></tr>
</table>

file dirname *fileName*	Returns directory path of *fileName*.
file executable *fileName*	Returns 1 if *fileName* is executable by user, 0 if not.
file exists *fileName*	Returns 1 if *fileName* exists (and user can read its directory), 0 if not.
file extension *fileName*	Returns all characters in *fileName* after and including the last dot.
file isdirectory *fileName*	Returns 1 if *fileName* is a directory, 0 if not.
file isfile *fileName*	Returns 1 if *fileName* is a regular file, 0 if not.
file join *name ?name ...?*	Joins file names using the correct path separator for the current platform.
file link *?-options? linkName ?target?*	(Tcl 8.4+) Creates a link from *linkName* to *target*. Returns link filename without *target*. Options are **-symbolic** or **-hard**.
file lstat *fileName varName*	Same as **file stat** except if *fileName* is a link, the status of the link is returned.
file mkdir *dirName ?dirName ...?*	Creates given directories with any needed parent directories. Trying to overwrite an existing file with a directory will result in an error. Args are processed in the order specified, halting at the first error, if any.
file mtime *fileName ?time?*	Returns the time that *fileName* was last modified as seconds since system epoch time. In Tcl 8.3+, *time* option sets last modified time.
file nativename *fileName*	Returns the platform-specific name of *fileName* .
file normalize *fileName*	(Tcl 8.4+) Returns a unique normalized (".." and "." are removed, symbolic links removed from dirname but not tail) file-system absolute path representation of *fileName* .
file owned *fileName*	Returns 1 if *fileName* owned by the user, 0 if not.
file pathtype *fileName*	Returns path type of fileName: **absolute** (specific file on a specific volume), **relative** (relative to the current working directory), or **volumerelative** (relative to the current working directory on a specified volume or specific file on the current working volume).
file readable *fileName*	Returns 1 if *fileName* is readable by user, 0 if not.
file readlink *fileName*	Returns target filename of symbolic link given by *fileName* or an error if fileName is not a link or can not be read.
file rename ?**-force** ? ?--? *source target*	Renames *source* file or directory to *target*, moving it if the target pathname specifies a name in another directory. The **-force** option forces overwriting of existing files.
file rename ?**-force** ? ?--? *source ?source ...? targetDir*	Moves each *source* file or directory to *targetDir* directory. Will not overwrite existing files unless **-force** is specified. Trying to overwrite a non-empty directory, overwrite a directory with a file, or a file with a directory will all result in errors. Args are processed in the order specified, halting at the first error, if any.
file rootname *fileName*	Returns all the characters in *fileName* up to but not including last dot (.").
file separator *?fileName?*	(Tcl 8.4+) Without arg returns the char used to separate path segments for native files on this platform. With arg does same for file system *fileName* is on.
file size *fileName*	Returns size of *fileName* in bytes.
file split *fileName*	Returns list whose elements are the path components of *fileName*.
file stat *fileName varName*	Place results of stat kernel call on *fileName* in array *varName* with elements **atime** (last accessed time), **ctime** (properties last updated time), **dev** (device),**gid** (group ID), **ino** (inode), **mode** (permissions),**mtime** (last modified time), **nlink** (number of hard links), **size** (total size in bytes),**type** (device type), and **uid** (user ID). All are decimal numbers except **type**, which is the same as **file type**. For links, returns status on linked to file.
file system *fileName*	(Tcl 8.4+) Returns a two element list for *fileName* with the name of file system and nature or type.

file tail *fileName*	Return all characters in *fileName* after last directory separator.
file type *fileName*	Returns type of *fileName*. Possible values are **file**, **directory**, **characterSpecial**, **blockSpecial**,**fifo**, **link**, or **socket**.
file volumes	Returns list of absolute paths of mounted volumes on system. Returns just "/" on UNIX, list of local drives on Windows, and list of local and network drives on MacOS.
file writable *fileName*	Returns 1 if *fileName* is writable by user, 0 if not.

2.09 History

When specifying an event to the history command, event may be either:

1. A number: if positive, it refers to the event with that number (all events are numbered starting at 1). If the number is negative, it selects an event relative to the current event (-1 refers to the previous event, -2 to the one before that, and so on). Event 0 refers to the current event.
2. A string: selects the most recent event that matches the string. An event is considered to match the string either if the string is the same as the first characters of the event, or match *pattern* using Pattern Globbing.

Command	Description
history	Same as **history info**.
history add *command ?***exec***?*	Adds *command* to history list, optionally executing it.
history change *newValue ?event?*	Replaces value of *event* (default is current) in history with *newValue*.
history clear	Erase the history list and reset event numbers.
history event *?event?*	Returns value of *event* (default is -1) in history.
history info *?count?*	Returns event number and contents of the last *count* events. Without *count* all events are returned.
history keep *?count?*	Set number of events to retain in history to count. Without *count*, returns current limit.
history nextId	Returns number for next event to be recorded in history.
history redo *?event?*	Re-evaluates event (default is -1).

Command Line Shortcuts

Syntax	Description
!!	Repeats the previous command
!n	Repeats command number n. If n is negative, it counts backward from the current command. The previous command is -1.
!prefix	Repeat the last command that starts with prefix.
!pattern	Repeat the last command that matches pattern.
^old^new	Replace all occurances of ols with new in the last command.

2.10 Input/Output

By default *channelID*s **stdin**, **stdout**, and **stderr** are open. These channels are not available on all platforms since they are not supported by the console.

<table>
<tr><td>Command</td><td>Description</td></tr>
<tr><td>cd ?dirName?</td><td>Change working directory to home directory or dirName if specified.</td></tr>
<tr><td>close channelId</td><td>Close the specified channelId. Will wait for child process(es) to complete for blocking channels. Will not return exit info for non-blocking channels.</td></tr>
<tr><td>eof channelID</td><td>Returns 1 if an end-of-file has occurred on channelID, 0 if not.</td></tr>
<tr><td>exec ?-keepnewline? ?--? arg ?arg ...?</td><td>Execute argsas subprocesses in a shell pipeline. Returns results to stdout of the last command in the pipeline unless redirected. Returns the error number, error message, stderr output (unless redirected), and sets errorCode (-errorcode return option for Tcl 8.5+) if a pipeline process is killed, suspended, exits abnormally, or writes to stderr without redirection. Also cleans up any pending children (detached PIDs). To retain the final newline char, use -keepnewline. Default stdin, stdout, and stderr are same as calling application. Performs "~" but not glob substitutions. The following args are used to redirect the I/O:
<table>
<tr><td>Redirection</td><td>Description</td><td>Redirection</td><td>Description</td></tr>
<tr><td>|</td><td>Pipe (stdout)</td><td>>> fileName</td><td>Append stdout to file</td></tr>
<tr><td>|&</td><td>Pipe (stdout and stderr)</td><td>2>> fileName</td><td>Append stderr to file</td></tr>
<tr><td>< fileName</td><td>Stdin from file</td><td>>>&fileName</td><td>Append stdout and stderr to file</td></tr>
<tr><td><@ channelID</td><td>Stdin from open file (UNIX only)</td><td>>@channelID</td><td>Stdout to open file (UNIX only)</td></tr>
<tr><td><< value</td><td>Pass value to stdin</td><td>2>@channelID</td><td>Stderr to open file (UNIX only)</td></tr>
<tr><td>> fileName</td><td>Stdout to file</td><td>>&@channelID</td><td>Stdout and stderr to open file (UNIX only)</td></tr>
<tr><td>2> fileName</td><td>Stderr to file</td><td>2>@1</td><td>(Tcl 8.4.7+) Redirects stderr to stdout</td></tr>
<tr><td>>& fileName</td><td>Stdout and stderr to file</td><td>&</td><td>Run in background. Returns list of pipeline PIDs.</td></tr>
<tr><td colspan="2"></td><td>// or \\</td><td>Refers to a network path</td></tr>
</table>
</td></tr>
<tr><td>fblocked channelID</td><td>Returns 1 if channelID does not have data available for reading, or 0 if it does.</td></tr>
<tr><td>fconfigure channelID ?option value ...? ?option value ...?</td><td>Sets and retrieves options for channelID. Sockets are read-only. Options are:</td></tr>
</table>

Option:	Type:	Description:
-blocking *boolean*	all	Whether I/O can block process. Default is to block. For MS Windows prior to Tcl 8.4, serial I/O always blocks.
-buffering *arg*	all	*Arg* is **full**, **line**, or **none** for buffer output. Default is full, except for channels that connect to terminal-like devices where its line. stdin and stdout are initially set to line, and stderr is set to none.
-buffersize *size*	all	Size of buffer in bytes. Range is 10 to 1,000,000 bytes. Default is 4096 bytes.
-encoding *name*	all	(Tcl 8.1+) Channel encoding. See Encodings. (ASCII, UNICODE, UTF-8, binary, etc.)
-eofchar *char*	all	Sets read EOF marker. \x1a for DOS.
-eofchar *{inChar outChar}*	all	Sets read and write EOF marker. No args returns a two element list with the current markers.
-error	all	(Tcl 8.0.5+) Returns last POSIX error message associated with channel or empty string if none.
-translation *mode*	all	Sets EOL marker. *Mode*s are **auto** (default is native newline), **binary** (no EOL), **cr**, **crlf**, and **lf**. Using binary implies -encoding binary.
-translation *{inMode outMode}*	all	Sets read and write EOLmarkers. *Mode*s are **auto** (default is native newline), **binary** (no EOL), **cr**, **crlf**, and **lf**. No args returns a two element list of in and out modes.
-peername	socket	For client or accepted sockets, returns a three element list with address, host name, and port number to which the peer socket is connected or bound
-sockname	socket	Returns a three element list with address, host name, and port number for the socket.
-handshake *type*	serial	(Tcl 8.4+ UNIX and MS Windows only) Setup automatic handshake control (none, rtscts, xonxoff, dtrdsr (MS Windows only)). Cannot be queried.
-lasterror	serial	(Tcl 8.3+ MS Windows) Returns a list of error details. Can only be queried.
-mode *baud, parity, data, stop*	serial	Set baud rate, *parity* (n, o, e, m, s), *data* bits (5 to 8), and *stop* bits (1 or 2) of channel.
-pollinterval *msec*	serial	(Tcl 8.2+ Windows) Max time between polling for fileevents. Default is 10 msec.
-queue	serial	(Tcl 8.4+ UNIX and MS Windows only) Returns a two element list of bytes in input and output buffers. Can only be queried.
-sysbuffer *inSize*	serial	(Tcl 8.4+ MS Windows) Change size of serial channel buffer. Default is 4096 bytes.
-sysbuffer *{inSize outSize}*	serial	(Tcl 8.4+ MS Windows) Change size of input and output serial channel buffers. Default is 4096 bytes.
-timeout *msec*	serial	(Tcl 8.4+ UNIX and MS Windows only) Set the timeout for blocking reads only. For Unix systems the granularity is 100 milliseconds.
-ttycontrol *{signal boolean signal boolean ...}*	serial	(Tcl 8.4+ UNIX and MS Windows only) Setup the handshake output lines or send BREAK. Cannot be queried.
-ttystatus	serial	(Tcl 8.4+ UNIX and MS Windows only) Returns a list of modem status and handshake input signals as a list of signal,value pairs.Can only be queried.
-xchar *{xonChar xoffChar}*	serial	(Tcl 8.4+ UNIX and MS Windows only) Query or change the software handshake chars. Default should be DC1 (0x11) (XON) and DC3 (0x13) (XOFF).

fcopy *inChID outChID* ?**-size** *size*? ?**-command** *callback* ?	Copy data from *inChID* to *outChID* until **eof** or *size* bytes are transferred. With **-command**, the copy runs in background and calls *callback* with args of bytes copied and an error message, if applicable, when done. Blocks without **-command**. In Tcl 8.4+, respects channel encodings.
fileevent *channelID option ?script?*	Create handler to evaluate *script* at global level when *channelID* becomes *option* (**readable** or **writable)**. Replaces the existing handler if present. The handler is deleted if *script* is an empty string, when the *channelID* is closed, or if the handler returns an error (**bgerror** will be called). *Script* needs to account for **eof**. Returns current script if *script* is not specified.

<table>
<tr><td>flush channelID</td><td>Flushes any output that has been buffered for channelID.</td></tr>
<tr><td>gets channelID ?varName?</td><td>Read the next line from channelID, discard the newline character, place the result in varName, and return the number of characters or -1 if there was an error. Without varName, the result is returned. Will return an empty string for non-blocking channels if no input is available.</td></tr>
<tr><td>glob ?option? ?--? pattern ?pattern ...?</td><td>Returns a list of all files in current directory that match any of the given csh-style glob patterns. See Pattern Globbing for expressions. Options are:</td></tr>
<tr><td> -directory directory</td><td>(Tcl 8.3+) Search for files in directory. Can not be used with -path .</td></tr>
<tr><td> -join</td><td>(Tcl 8.3+) Join pattern args into a single pattern with directory separators.</td></tr>
<tr><td> -nocomplain</td><td>Allows an empty list to be returned without error.</td></tr>
<tr><td> -path pathPrefix</td><td>(Tcl 8.3+) Search for files starting with pathPrefix. Can not be used with -directory.</td></tr>
<tr><td> -tails</td><td>(Tcl 8.4+) Only return filename and not path when used with -path or -directory .</td></tr>
<tr><td> -types typeList</td><td>(Tcl 8.3+) Only list items which match types in typeList. The first form shows matches of one or more of the following types: b (block special file), c (character special file), d (directory), f (plain file), l (symbolic link), p (named pipe), or s (socket). The second form only shows matches of all the specified types. The available types are: r (read), w (write), x (execute), readonly, hidden, or the MacOS type. The second form may also use types from the first form.</td></tr>
<tr><td>open fileName ?access? ?perms?</td><td>Opens a file, serial port, or command pipeline and returns its channel ID. If the first char of fileName is "|" then fileName is opened as a pipeline process with the same redirection options as exec. If filename is a serial port, then the specified port is used (/dev/ttyX (X=a or b) on UNIX and com#: (#=1 to 4) on Windows). If a new file is created, its permission are set to perms (default is 0666) in conjunction with processes umask. A pipeline with w access writes to stdout unless redirected. A pipeline with r access reads from stdin unless redirected. The access options are:
<table>
<tr><th>UNIX</th><th>Description</th></tr>
<tr><td>r</td><td>Read only (default). FileName must exist.</td></tr>
<tr><td>r+</td><td>Read/write. FileName must exist.</td></tr>
<tr><td>w</td><td>Write only. Truncate fileName, if exists.</td></tr>
<tr><td>w+</td><td>Read/write. Truncate fileName, if exists.</td></tr>
<tr><td>a</td><td>Write only. Set access position to end.</td></tr>
<tr><td>a+</td><td>Read/write. Set access position to end.</td></tr>
</table>
<table>
<tr><th>POSIX</th><th>Description</th></tr>
<tr><td>RDONLY</td><td>Read only</td></tr>
<tr><td>RDWR</td><td>Read/write.</td></tr>
<tr><td>WRONLY</td><td>Write only.</td></tr>
<tr><td>APPEND</td><td>Set access position to end for each write.</td></tr>
<tr><td>CREAT</td><td>Create fileName if it doesn’t exist.</td></tr>
<tr><td>EXCL</td><td>Used with CREAT, fileName must not exist.</td></tr>
<tr><td>NOCTTY</td><td>Prevent terminal device from being the controlling terminal.</td></tr>
<tr><td>NONBLOCK</td><td>Do not block during opening.</td></tr>
<tr><td>TRUNC</td><td>TruncatefileName if it exists.</td></tr>
</table></td></tr>
<tr><td>pid ?channelID?</td><td>Return a list of process IDs, in order, for pipeline process channelID. Without channelID, returns process ID of interpreter process.</td></tr>
<tr><td>puts ?-nonewline ? ?channelID? string</td><td>Write string to channelID (default is stdout). Omit newline with -nonewline. Newline is based on fconfigure -translation for channelID.</td></tr>
<tr><td>pwd</td><td>Returns the current working directory. Guaranteed to be the unique normalized string representation of the path in Tcl 8.4+.</td></tr>
<tr><td>read ?-nonewline? ?channelID?</td><td>Read all remaining data from channelID, optionally discarding last character if it is a newline.</td></tr>
<tr><td>read channelID numChars</td><td>Read numChars (byte size depends on encoding) or remaining if less available from channelID. For serial ports, if numChars is not specified will read until EOF.</td></tr>
<tr><td>seek channelID offset ?origin?</td><td>Change current access position for channelID to offset bytes from origin. Origin options are: start (default), current, or end.</td></tr>
</table>

socket *?option ...? host port*	Open a read/write client-side TCP socket to server *host* on *port* and returns the channel ID. The local *host* can be specified with **localhost**. Options are:
-async	Make connection asynchronous.
-myaddr *addr*	Set network address of client (if multiple available). Default is system specific.
-myport *port*	Set connection port of client (if different from server). Default is random port.
socket -server *command ?option? port*	Open server TCP socket on *port*. For each connection made, invoke *command* with three args: the channel, client address, and client port number. If *port* is 0, the OS will use an unassigned port.
-myaddr *addr*	Sets the network address of server to *addr*.
tell *channelID*	Returns current access position for *channelID* in bytes.

2.11 Interpreter Information

Command	Description
info args *procName*	Returns list with names of arguments to procedure *procName*.
info body *procName*	Returns the body of procedure *procName*.
info cmdcount	Returns the total number of commands that have been invoked in this interpeter.
info commands *?pattern?*	Returns list of all Tcl commands (built-ins and procs) in current namespace or those matching *pattern* using Pattern Globbing.
info complete *command*	Returns 1 if command is a complete Tcl command, 0 if not. Complete means having no unclosed quotes, braces, brackets or array element names.
info default *procName arg varName*	Returns 1 if procedure *procName* has a default for argument *arg* and places the value in variable *varName*. Returns 0 if there is no default.
info exists *varName*	Returns 1 if the variable *varName* exists in the current context, 0 if not.
info functions *?pattern?*	(Tcl 8.4+) Returns list of all math functions or those matching *pattern* using Pattern Globbing .
info globals *?pattern?*	Returns list of all global variables or those matching *pattern* using Pattern Globbing.
info hostname	Returns name of computer on which interpreter was invoked.
info level *?number?*	Returns the invoking procedure stack level or if number is specified, a list of the name and args of procedure call at level *number* on the stack. info level 0 returns the curent proc name and args.
info library	Returns name of library directory where standard Tcl scripts are stored. Same as variable **TCL_LIBRARY**.
info loaded *?interp?*	Returns list of all packages loaded or just those in *interp* if specified. Each list element consists of the source filename and package name.
info locals *?pattern?*	Returns list of all local variables or those matching *pattern* using Pattern Globbing.
info nameofexecutable	Returns full pathname of binary from which the application was invoked.
info patchlevel	Returns current patch level for Tcl. Same as variable **tcl_patchLevel**.
info procs *?pattern?*	Returns list of all Tcl procedures in current namespace or those matching *pattern* using Pattern Globbing.
info script *?filename?*	Returns name of Tcl script currently being evaluated (by source), if any. In Tcl 8.4+, if *filename* is specified, the return value of **info script** is set to *filename*.
info sharedlibextension	Returns extension used by platform for shared objects.
info tclversion	Returns version number of Tcl in major.minor form. Same as variable **tcl_version**.
info vars *?pattern?*	Returns list of all currently-visible variables or those matching *pattern* using Pattern Globbing.
memory *option ?arg arg ...?*	(Tcl 8.4+) Allows control of the Tcl memory debugging capabilities. Tcl must be compiled with memory debugging enabled. Options are:
active file	Output a list of all currently allocated memory (with associated tags) to file.
break_on_malloc count	After count allocations, Tcl will output a break message and SIGINT to the C debugger.
info	Reurns the total number of allocations and frees, current packets allocated, current bytes allocated, and the maximum number of packets and bytes allocated.
init fn	Turn on or off the pre-initialization of all allocated memory with bogus bytes.
onexit file	Output a list of all currently allocated memory (with associated tags)to file at Tcl exit.
tag string	Sets the tag value to string for subsequent calls to ckalloc.
trace fn	Turn on or off the output to stderr of memory tracing info. Each ckalloc or ckfree outputs: fn, address, size, C filename of calling procedure, and line in file.
trace_on_at_malloc count	After count allocations, Tcl will enable memory tracing.
validate fn	Turn on or off memory validation (if the ckalloc or free overwrite another allocated portion of memory).

2.12 Interpreters

Command	Description
interp alias *srcPath srcCmd*	Returns list whose elements are the targetCmd and args associated with the alias *srcCmd* in interpreter *srcPath*.
interp alias *srcPath srcCmd {}*	Deletes the alias *srcCmd* in interpreter *srcPath*.
interp alias *srcPath srcCmd targetPath targetCmd ?arg ...?*	Creates an alias *srcCmd* in interpreter *srcPath* which when invoked will run *targetCmd* and *args* in the interpreter *targetPath* . In *targetPath* , the current interpeter is {}.
interp aliases *?path?*	Returns a list of all alias source commands defined in the interpreter identified by *path* .
interp bgerror path ?cmdPrefix?	(Tcl 8.5+) Sets the command (in list format) to handle background errors in the path interp. Without cmdPrefix, the currently registered command, if any, or the background error handler (defined by bgerror), will be returned. When an error occurs in path interp and it cannot be reported up the procedure stack, the returned error message and dictionary of return options (see catch) will be appended to cmdPrefix and the new command will be evaluated by the Tcl Interpreter.
interp create *?***-safe** *? ?--? ?path?*	Creates a slave interpreter (optionally safe) identified by *path* with a slave name obtained by removing the last component from *path* .
interp delete *?path ...?*	Deletes the interpreters defined by the *path* args and all their slave interpreters.
interp eval *path arg ?arg ...?*	Evalutes concatenation of *arg* s as a command in interpreter *path* .
interp exists *path*	Returns 1 if interpreter *path* exists, 0 if not.
interp expose *path hiddenCmd ?exposedCmdName?*	Make *hiddenCmd* in interp*path* exposed (optionally as *exposedCmd Name*).
interp hide *path exposedCmdName ?hiddenCmdName?*	Make *exposedCmd* in interp*path* hidden (optionally as *hiddenCmd Name*).
interp hidden *path*	Returns list of hidden commands in interp *path*.
interp invokehidden *path ?options? ?--? hiddenCmdName ?arg ...?*	Invokes *hiddenCmdName* with specified args in interp *path* at the current call frame and can access local variables in that and outer call frames.
-global	Invokes hidden command at the global level in the target interpreter. Overrides -namespace.
-namespace *namespace*	(Tcl 8.5+) Invokes hidden command in the specified namespace in the target interpreter
interp limit path limitType ?option? ?value ...?	(Tcl 8.5+) Set or query the resource limit limitType for the interp path. Without value, the current value is returned. Without option, the current config of limitType is returned. The two kinds of limitTypes, command and time. Command restricts the total the total number of Tcl commands that may be executed by an interpreter (using info cmdcount) and time limits the total execution time (using time) of the interpreter. When the limit for an interpreter is exceeded, the -command callback is evaluated, if defined. If the limit is still in force, an error is generated to the interpreter’s invoking command. Valid options are:

-command cmd ?arg ...?	Specifis the Tcl script to eval in the global namespace of the interpreter reading and writing the option when the particular limit in the limited interpreter is exceeded. The callback may modify the limit to allow the interpreter to continue executing. If the callback generates an error, it is reported through the background error mechansism (see interp bgerror or bgerror)
-granularity	
-milliseconds	
-seconds	
-value	
interp issafe *?path?*	Returns 1 if interpreter *path* is safe, 0 if not.
interp marktrusted *?path?*	Marks interp *path* as trusted. Does not expose the hidden commands.
interp recursionlimit *path ?newlimit?*	(Tcl 8.4+) Returns the max allowable nesting depth for the interpreter *path* . If *newlimit* is defined, the recursion limit is set to it.
interp share *srcPath channelID destPath*	Sets the I/O channel *channelID* in interpreter *srcPath* to be shared with interpreter *destPath* .
interp slaves *?path?*	Returns list of names of all slave interpreters of interpreter *path*. If path is omitted, the invoking interpreter is used.
interp target *path alias*	Returns list describing target interpreter of *alias* in interpreter *path*.
interp transfer *srcPath channelID destPath*	Moves I/O channel *channelID* from interpreter *srcPath* to *destPath*.
::safe::interpCreate ?*slave*? ?*option arg...*?	Creates a safe interpreter, installs the specified aliases, and initializes the auto-loading and package mechanism. Without *slave*, returns the interpreter name.
-accessPath *directoryList*	Sets the list of directories from which the safe interpreter can source and load files. For the default option or if set to an empty list, the safe interpreter will use the same directories as its master for auto-loading.
-statics *boolean*	(Tcl 8.0p1+) Specifies if the safe interpreter will be allowed to load statically linked packages. Default is true.
-noStatics	Convenience shortcut for **-statics false**.
-nested *boolean*	(Tcl 8.0p1+) Specifies if the safe interpreter will be allowed to load packages into its own sub-interpreters. Default is false.
-nestedLoadOk	convenience shortcut for **-nested true**.
-deleteHook *script*	Evaluate *script* in the master just before deleting a safe interpreter. Passes name of slave interpreter as arg to script. For the default option or if set to an empty string, the current script is removed for current safe interpreter.
::safe::interpInit *slave* ?*option arg...*?	Similar to **interpCreate** except it that does not create the safe interpreter. *slave* must have been created by some other means, like **interp create -safe**. Uses same options as **::safe::interpCreate**.
::safe::interpConfigure *slaveoption arg ...*? ?	Sets *option* to specified *arg* for interpreter *slave*. Without args, returns setting for*option*. Without *options*, returns current interpreter settings. Uses same options as **::safe::interpCreate**.
::safe::interpDelete *slave*	Deletes the safe interpreter *slave* .
::safe::interpAddToAccessPath *slave directory*	Adds *directory* to the virtual path maintained for the safe interpreter *slave* (if not already in the path), and returns the token that can be used in the safe interpreter to obtain access to files in that directory.
::safe::interpFindInAccessPath *slave directory*	This command finds and returns 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::loadTk *slave* ?**-use** *windowId*? ?**-display** *displayName* ?	Load Tk into a safe interpreter.*WindowId* identifies the window on *displayName* to contain the "." window of the interpreter.
::safe::setLogCmd ?*cmd arg...*?	Installs a script that will be called when interesting life cycle events occur for a safe interpreter. Calls script with text message arg added to describe the event. If *cmd* is set to an emptry string, the currentlt installed script is removed and logging is turned off. Without *cmd* and *args*, returns currently installed script.

Slave Interpreters

For each slave interpreter created with the interp command, a new Tcl command is created in the master interpreter with the same name as the new interpreter. This command may be used to invoke various operations on the interpreter. The following commands are used like interp, but without the srcPath or path (defaults to the slave itself) and the targetPath arguments (defaults to the slave's master).

alias	bgerror	expose	hidden	issafe	marktrusted
aliases	eval	hide	invokehidden	limit	recursionlimit

Safe Interpreter Exposed Commands

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

Safe Interpreter Hidden Commands

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

Tcl Library Commands Not Included in a Safe Interpreter

auto_exec_ok	**auto_load**	**auto_qualify**
auto_import	**auto_load_index**	**unknown**

Auto Loaded Commands Not Included in a Safe Interpreter

Without the unknown command, the default loading facilities are not available. The following commands are normally autoloaded:

auto_mkindex	**::safe::interpAddToAccessPath**	**tcl_endOfWord**
auto_mkindex_old	**::safe::interpCreate**	**tcl_findLibrary**
auto_reset	**::safe::interpConfigure**	**tcl_startOfNextWord**
history	**::safe::interpDelete**	**tcl_startOfPreviousWord**
parray	**::safe::interpFindInAccessPath**	**tcl_wordBreakAfter**
pkg_mkIndex	**::safe::interpInit**	**tcl_wordBreakBefore**
::pkg::create	**::safe::setLogC**	

Safe Interpreter Aliases

Command	Description
source *fileName*	Sources *fileName* into the safe interpreter. Only files in directories included in the virtual path for the safe interpreter can be used. Requires the safe interpreter to use one of the token names in its virtual path to denote the directory in which the file to be sourced can be found.
load *fileName*	Loads a shared object file *fileName* into the safe interpreter. The filename must contain a token name mentioned in the virtual path for the safe interpreter for it to be found successfully. The shared object file must contain a safe entry point.
file *?subCmd args ...?*	Provides access to a safe subset of the subcommands of the **file** command. Only **dirname**, **join** ,**extension**,**root**,**tail**,**pathname**, and **split** subcommands are accessible.
encoding *?subCmd args ...?*	Provides access to a safe subset of the **encoding**subcommands. The system encoding cannot be changed, but all other subcommands are accessible.
exit	The calling interpreter is deleted and its computation is stopped, but the Tcl process in which this interpreter exists is not terminated.

2.13 Lists

A special case of string which consists of a series of elements which can be indexed like an array starting with 0. Elements may contain strings, numbers, etc. If spaces or other special characters are used they must grouped within braces or use backslash substitution as required. Elements may consist of nested sublists, which can contain more sub-lists, etc. to any depth. Common definitions for strings as lists:

Definition	Input	Criteria	Result
Well Formed List	string *s*	**string equal "{**$*s*}" **[list** $*s*]	returns 1
Canonical List	well formed list *s*	**string equal** $*s* **[split**$*s*]	returns 1
Nested List	list [list 1a 1b] [list 2a 2b] ...		

The list arguments of *index*, *first*, and *last* can be replaced with **end** to use the index of the last element in *list* . For the list commands below, *list* is the contents of the list (use variable substitution, i.e. $*listName*) and *listName* is the name of the list.

Command	Description
concat *?arg arg ...?*	Returns concatenation of *arg*s into a single list while trimming leading and trailing spaces. Removes one level of grouping before forming list.
join *list ?joinString?*	Returns string created by joining all elements of *list* with *joinString* (default is space) separating each element.
lappend *listName ?value ...?*	Appends each *value* arg to the end of list *listName*.

lassign list varName ?varName ...?	(Tcl 8.5+) Assigns successive elements from list to the variables given by the varName args and returns unassigned list elements. If ther are more var args then list elements, unused vars are set to {}.
lindex *list ?index ...?*	Returns value of element at *index* in *list*. In Tcl 8.1.1+, *index* can be **end-***#* (where *#* is an integer) for the last element minus the specified number. In Tcl 8.4+, multiple indicies may be used (in list or indivdual args format) for sub-lists (nested list) of *list* in highest to lowest depth order. Without *index*, returns contents of *list*. Invalid indicies return {}.
linsert *list index element ?element ...?*	Returns new list by inserting *element*s at *index* in *list*. Inserts at beginning of list for *index* <= 0 and at end of list for *index* of **end** or *index* > elements in list. In Tcl 8.1.1+, *index* can be **end-***#* (where *#* is an integer) for the last element minus the specified number.
list *?arg arg ...?*	Returns new list formed by using each *arg* as an element. Does not alter grouping. Prior to Tcl 8.5, list does not quote leading #'s in an eval safe manner.
llength *list*	Returns number of elements in *list*.
lrange *list first last*	Returns new list consiting of *list* elements *first* through *last*, inclusive. If *first* <= 0 then 0 is used and if *last* > elements in list, then **end** is used. Returns empty list if *first* > *last*. In Tcl 8.1.1+, *first* and last can be **end-***#* (where *#* is an integer) for the last element minus the specified number.
lrepeat number element ?element ...?	(Tcl 8.5+) Creates a list of elements repeated number of times where number is >= 0.
lreplace *list first last ?element ...?*	Returns new list formed by replacing elements *first* through *last* in *list* with *element*s. If *first* <= 0 then 0 is used and if *last* > elements in list, then **end** is used. If *first* < *last* < 0, then new elements will be prepended to the list. If *first* > *last*, new elements are inserted before *first*. Without *element* args, elements between *first* and *last* are deleted. If *list* is empty, *elements* are added to the end of the list. In Tcl 8.1.1+, *first* and last can be **end-***#* (where *#* is an integer) for the last element minus the specified number.
lsearch *?options? list pattern*	Returns index of first element in *list* that matches *pattern* (-1 for no match). Mutually exclusive options where last specified is used: **-exact**, **-glob**, **-regexp** , and **-sorted**; **-ascii**, **-dictionary**, **-integer** , and **-real**; **-increasing** and **-decreasing**. Options are:
-all	(Tcl 8.4+) Returns list of all matching indices or all matching values if used with **-inline**. Returns empty list if no matches are found.
-ascii	(Tcl 8.4+) Compare to elements as ASCII strings (alphabetical, case sensitive). Used with **-exact** or **-sorted**.
-decreasing	(Tcl 8.4+) Elements are in decreasing order. Used with **-sorted**.
-dictionary	(Tcl 8.4+) Compare to elements using dictionary-style (alphabetical, case insensitive) comparisons. Used with **-exact** or **-sorted**.
-exact	The list element must contain exactly the same string as *pattern*.
-glob	Compare to elements using Pattern Globbing. This is the default option.
-increasing	(Tcl 8.4+) Elements are in increasing order. Used with **-sorted**.
-index indexList	(Tcl 8.5+) Treat list elements as sublists (nested lists) and only searches in the sub-element specified by indexList in highest to lowest depth order.
-inline	(Tcl 8.4+) Returns matching value instead of index or empty string if no match. If used with **-all** , a list of all matched values is returned.
-integer	(Tcl 8.4+) Compare to elements as integers (numeric). Used with **-exact** or **-sorted**.
-not	(Tcl 8.4+) Negates match criteria. Index of first non-matching element will be returned.
-real	(Tcl 8.4+) Compare to elements as floating point values. Used with **-exact** or **-sorted.**
-regexp	Compare to elements using Regular Expression *pattern* matching. Prepend (?i) to exp for case insenstive.
-sorted	(Tcl 8.4+) Specifies that the list elements are in sorted order, so use a more efficient search algorithm. Default options are: **-exact**, **-ascii**, and **-increasing** . Can not be used with **-all**, **-glob**, **-not** or **-regexp** .
-start *index*	(Tcl 8.4+) Start search at *index* .
-subindicies	(Tcl 8.5+) Used with -index to return a listof subindicies for the matching element in highest to lowest depth order.

lset *listName ?index...? newValue*	(TCL 8.4+) Replaces element at *index* in list *listName* with *newValue* and returns the new list. Without *index* replaces all of *list* with *newValue*. *Index* can be **end-***#* (where # is an integer) for the last element minus the specified number.Multiple indicies may be used (in list or indivdual args format) for sub-lists (nested list) of *list* in highest to lowest depth order. An error is returned if *index* < 0 or *index* > number of elements in list.
lsort *?options? list*	Returns new list formed by sorting *list* according to options:
-ascii	Sort elements in ASCII order (alphabetical, case sensitive). (default)
-command *cmd arg arg*	Use *cmd* to compare two *arg*s of elements where *cmd* returns an integer <, =, or > than 0 to denote corresponding compare result.
-decreasing	Sort elements in decreasing order.
-dictionary	Sort elements using dictionary-style (alphabetical, case insensitive) order. Sorts numbers as integers not chars, but in ascending absolute value order.
-increasing	Sort elements in increasing order. (default)
-indices	(Tcl 8.5+) Returns the indices of the given list's elements in the order that they would have otherwise been sorted.
-index *index*List	Treat list elements as sublists (nested lists) and sorts based on the element at *index*List in the sub-list. In Tcl 8.1.1+, *index*List can be **end-***#* (where # an integer) for the last element minus the specified number. In Tcl 8.5+, indexList is a list of sub-indicies, in highest to lowest depth order, specifying the sub-elements to be used for the sort**.**
-integer	Converts elements to integers and sorts in numeric order. Can not sort any number containing a decimal point or exponent. Binary data is not allowed.
-real	Converts elements to floating-point values and sorts in numeric order.
-unique	(Tcl 8.3+) Retain only the last set of duplicate elements.
split *string ?splitChars?*	Returns a list formed by splitting *string* at each character in *splitChars* (default is white-space [\t\n\r]). The *splitChars* will not be included in the new list. Empty list elements will be created when multiple *splitChars* appear next to each other or at the start or end of *string*. If *splitChars* is an empty string each char in *string* will become a separate list element.

2.14 Namespaces

Namespaces are used to partition a collection commands and variables from another collection so they don't interfere with each other. Namespace variables resemble global variables in Tcl. They exist outside of the procedures in a namespace but can be accessed in a procedure via the **variable** command. Namespaces are denoted by *namespace* **::** *variable* where *variable* can be a nested *namespace* and *variable* such as *namespace* **::** *namespace* **::***variable*. The global namespace holds all global variables and commands. The global namespace is "" (empty string) but is denoted by **::** .

Command	Description
auto_qualify *cmd namespace*	(Tcl 8.0p1+) Computes a list of fully qualified names for *cmd* in *namespace* then the global namespace.
namespace children *?namespace? ?pattern?*	Returns list of all child namespaces belonging to *namespace* (default is current) or match *pattern* using Pattern Globbing.
namespace code *script*	Returns a new script, that when evaluated will cause *script* to be evaluated in the current (where **namespace code** was invoked) namespace. Useful for callbacks. Additional args can be passed to *script* by appending to the new script when evaluated.
namespace current	Returns fully-qualified name of current namespace.
namespace delete *?namespace ...?*	Each *namespace* is deleted along with any child namespaces, procedures, and variables. If a procedure is currently executing in *namespace*, it will be deleted when the procedure returns.
namespace ensemble create ?option value ...?	(Tcl 8.5+) Creates a new ensemble command linked to the current namespace, returning the fully qualified name of the command created. Valid options are:

-command command ?arg...?	Specifies name of ensemble command. Default is to create an ensemble with exactly the same name as the linked namespace.
-map dict	Specifies a dictionary to use for mapping from subcommand names to a list of prefix words to use in place of the ensemble command and subcommand words. Default is to map from the local name of the subcommand to its fully-qualified name.
-prefixes boolean	Specifies whether the ensemble command recognizes unambiguous prefixes of its subcommands (default) or only exact matches.
-subcommands list	Specifies valid ensemble subcommands. Default is to use the keys of the dictionary per the -map option or the exported commands of the linked namespace at the time of the invocation of the ensemble command.
-unknown command ?arg...?	Specifies the command to append unknown ensemble sub-commands and eval in the scope of the attempted call. Default is to generate an error like Tcl_GetIndexFromObj. The command must return either a list of command words to replace the ensemble command and subcommand like -map, or an empty list. The latter will result in an attempt to eval the ensemble sub-command again and if unsuccessful will generate an error like the default case.
namespace ensemble configure command ?option? ?value ...?	(Tcl 8.5+) Change the ensemble *option* to *value*. Without *value*, the current value is returned. Without *option*, a list of all available options for the ensemble is returned. Valid options are:
-map dict	Specifies a dictionary to use for mapping from subcommand names to a list of prefix words to use in place of the ensemble command and subcommand words. Default is to map from the local name of the subcommand to its fully-qualified name.
-namespace	Returns the fully-qualified name of the namespace in which the ensemble was created.
-prefixes boolean	Specifies whether the ensemble command recognizes unambiguous prefixes of its subcommands (default) or only exact matches.
-subcommands list	Specifies valid ensemble subcommands. Default is to use the keys of the dictionary per the -map option or the exported commands of the linked namespace at the time of the invocation of the ensemble command.
-unknown command ?arg...?	Specifies the command to append unknown ensemble sub-commands and eval in the scope of the attempted call. Default is to generate an error like Tcl_GetIndexFromObj. The command must return either a list of command words to replace the ensemble command and subcommand like -map, or an empty list. The latter will result in an attempt to eval the ensemble sub-command again and if unsuccessful will generate an error like the default case.
namespace ensembleexists command	(Tcl 8.5+) Returns 1 if command exists and is an ensemble, otherwise returns 0.
namespace eval *namespace arg ?arg ...?*	Activates *namespace* and evaluates concatenation of *arg*s inside it. Counts as a level for **uplevel** and **upvar**.
namespace exists *namespace*	(Tcl 8.4+) Returns 1 if *namespace* is valid in current context, 0 if not.
namespace export *?***-clear***? ?pattern...?*	Appends commands matching *pattern* (without namepsace qualifiers)using Pattern Globbing to export list of current namespace. If **-clear** is given, the export list is first emptied. Without any args, the current namespace's export list is returned.
namespace forget *?pattern ...? ?namespace::pattern ...?*	Removes from current namespace any previously imported commands matching *pattern* using Pattern Globbing or from exported namespace *namespace.*
namespace import *?***-force***? ?namespace::pattern...?*	Imports into current namespace commands matching *pattern* using Pattern Globbing from *namespace* . The **-force** option allows replacing of existing commands.
namespace inscope *namespace listArg ?arg ...?*	Activates *namespace* (which must already exist) and evaluates inside it the result of **lappend** *listArg arg*s.
namespace origin *command*	Returns fully-qualified name of original command that imported *command* refers to.
namespace parent *?namespace?*	Returns fully-qualified name of parent namespace for *namespace*. Without *namespace*, returns parent of current namespace.
namespace qualifiers *string*	Returns any leading namespace qualifiers for *string*.
namespace tail *string*	Returns the simple name (without leading namespace qualifiers) for string.

namespace which ?**-command**? ?**-variable**?*name*	Returns fully-qualified name of the command (default) or variable (if **-variable** used) *name* in the current namespace. Will look in global namespace if not in current namespace. Returns empty string if doesn't exist.
variable *name ?value? ?name value ...?*	Creates *name* variables in current namespace (if unqualified) initialized to *value* (default is to leave undefined for new vars or current value for existing vars). *Name* can reference an array but not an element in an array. In this case *value* should not be used. Used inside a procedure but outside of an **namespace eval**, a local variable is created linked to the given namespace variable.

2.15 Packages

Packages are used to partition subroutines or entire programs into portable packages that can be used in other applications or subroutines. Each package can contain a version in Major.Minor?.subreleases...? format where only versions with the same major version are assumed to be compatible. Packages are indexed using the pkg_mkIndex command.

Command	Description
package forget *package ?package...?*	Removes all info about packages from interpreter.
package ifneeded *package version ?script?*	Tells interpreter that evaluating *script* will provide *version* of *package*. Without *script*, current script for *version* of *package* is returned or empty string if none.
package names	Returns list of all packages in the interpreter that are currently provided or have an **ifneeded** script available.
package present ?**-exact**? *package ?version?*	(Tcl 8.1+) Same as **package require** except does not try to load package if not already loaded.
package provide *package ?version?*	Tells interpreter that *version* of *package* is now present. Without version, the currently provided version of *package* is returned or empty string if none.
package require ?**-exact**? *package ?version?*	Tells interpreter that *version* of *package* is needed. Only packages with versions equal to or later than *version* (if provided) are acceptable, but must have same major version. If **-exact** is specified, the exact *version* specified must be provided. Without *version* or **-exact**, any version is acceptable. Returns version number loaded.
package unknown *?command?*	Specifies *command* to invoke for **package require** if a suitable version of package can not be found in **package ifneeded** database. With *command*, Tcl appends two args for the package name and version when invoked or removes **package unknown** if its an empty string. Without *command*, the current **package unknown** script is returned.
package vcompare *version1 version2*	Returns -1 if *version1* is earlier than *version2*, 0 if equal, and 1 if later.
package versions *package*	Returns list of all versions numbers of *package* in **package ifneeded** database.
package vsatisfies *version1 version2*	Returns 1 if *version2* scripts will work unchanged under *version1* (version1 >= version2 and both samesame major version #), 0 if not.
::pkg::create -name *packageName* **-version** *packageVersion* ?**-load** *filespec*? ... ?**-source** *filespec*? ...	Construct an appropriate **package ifneeded** command for *packageName*. Where **-load** is used with **load** cmd and *filespec* is a two item list of filename and a list of cmds provided. **-source** is used with the **source** cmd.
pkg_mkIndex *?options? ?--? directory ?pattern ...?*	Creates the pkgIndex.tcl file in the specified *directory* with all files matching *pattern* using Pattern Globbing.
-direct	(Tcl 8.0.4+) A packages in index file will be loaded upon **package require**. (default in Tcl 8.3+)
-lazy	(Tcl 8.3+) A package in index file will be loaded when one of the provided commands is used. (default up to Tcl 8.2.3)
-load *pkgPat*	(Tcl 8.0.4+) Packages that match *pkgPat* using Pattern Globbing in the current interpreter will be pre-loaded into slave interpreter used to generate index. In Tcl 8.4.2+ match is case insensitive.
-verbose	(Tcl 8.0.4+) Generate output to stderr during indexing process.

2.16 Procedures

Procedures are used to partition segments of code into subroutines so they can be called from other parts of an application or recursively. Procedures behave just like built-in commands and can have variable length arg lists. Variables within a procedure can be decalared as local (default) or global.

Command	Description
name ?args?	Calls procedure *name* with optional *args*.
global *varName ?varName ...?*	Creates local variables (result of namespace tail) linked to the global or namespace qualified variables varName. Only valid within procedures. VarName can reference an array but not an element in an array. Tcl 8.5+ will return an error for array elements.
proc *name {arg ?default?...} {body}*	Create a new Tcl procedure named *name* (or replaces existing procedure) where *arg*s is a list of arguments (each element is list of arg name and optional default value) and *body* is Tcl commmands to evaluate when invoked. *Name* can contain namespace qualifiers. If args is used as the last arg, all remaining args will be combined into a list and assigned to the args variable. Don't use "." as a proc name with Tk.
return ?options? ?*string*?	Return immediately from current procedure, top-level command, or **source** command with *string* (default is empty string) as the returned value. Options are:
-code *code*	Valid return codes are: **ok** (0), **error** (1), **return** (2), **break** (3), **continue** (4), or an integer.
-errorcode *error*	Used with -code error, to set the global variable **errorCode** to *error*. Used for additional info about the error (in list format for Tcl 8.5+ and defaults to NONE).
-errorinfo *info*	Used with -code error, to set the global variable **errorInfo** to *info*. Used for the procedure stack trace (in list format for Tcl 8.5+).
-level level	(Tcl 8.5+) Number of levels (default is 1) up the calling stack to return code to (intermediate steps get code return).
-options options	(Tcl 8.5+) Dictionary of options to return.
uplevel	(See Variables)
upvar	(See Variables)

2.17 Strings

A string is an arbitrary series of bytes (including binary data with null characeters) of any size up to the amount of available virtual memory. Character Strings are a special type of string kept in UTF-8 encoding by Tcl. Most Tcl commands expect to work on character strings and may not be able to handle binary data. Each character in a string is indexed like an array starting with index 0. The string command arguments of *index , startIndex, charIndex, lastIndex, first,* and *last* can be replaced with **end** to use the index of the last character in *string* . In Tcl 8.1.1+, **end-***number* (where *number* an integer) can be used to specify an index of the last character minus the specified *number*.

Command	Description
append *varName ?value ...?*	Appends each of the given *value*s to the string stored in *varName*.

binary format *formatString ?arg ...?*

Returns the *args* converted to a binary string based on *formatString*. The *formatString* is a sequence of field specifiers and optional integer count pairs separated by 0 or more spaces. The default count is 1. For strings and positions the count is the size and a count of "*" indicates all bytes/chars in *arg* will be used, otherwise will truncate if too long or pad if too sort. For ints and floating points it is the number of repetitions. Binary and hex types zero pad to the byte boundary if count > num of bytes/chars or truncate if count < num of bytes/chars. The field specifiers are:

String Types	
a	ISO 8859-1 chars (8-bit, null pad)
A	ISO 8859-1 chars (8-bit, space pad)
Position Types	
x	Null (\0)
X	Backspace (X* skip to start)
@	Absolute position (@* skip to end)

Type	Size (bits)	Native	Little Endian	Big Endian
Binary (0 pad, byte boundary)	1		b	B
Hex (0 pad)	4		h	H
Char	8	c	c	c
Short Int	16	t (8.5+)	s	S
Int	32	n (8.5+)	i	I
Wide Int	64	m (8.5+)	w (8.4+)	W (8.4+)
Float (IEEE)	32	f	r (8.5+)	R (8.5+)
Double (IEEE)	64	d	q (8.5+)	Q (8.5+)

binary scan *string formatString varName ?varName ...?*

Converts binary data into *varName* string variables based on *formatString*. Returns the number of strings converted. Stores integers as signed ints. The format field specifiers are the same as **binary format** except for:

a	ISO 8859-1 chars (no pad stripping)	**A**	ISO 8859-1 chars (strip null & space pad)	**x**	skip

format *formatString ?arg ...?*

Returns a formated string similar to the ANSI C sprintf. The format string is *%[argpos$][flag][width][.prec][len]char* where argpos, width, and prec are integers. Fields are:

Field	Description
argpos	Specifies arg to use for value with format {arg #}$. Argpos can be a variable by using ${var}$ or if quoting "${var}\$". Uses successive args for * specifiers. If any positional specifier is used, then all conversions must use them. Optional field.
flag	Conversion flag. See options below. Optional field.
width	Minimum field width. Uses *flag* specified padding. A field width of * uses the next arg as the field size. Optional field.
.prec	Value is decimal places for **e**, **E**, and **f**; total digits for **g** and **G**; trunc size for ints; and max chars for **s**. If precision is *, the next arg is used as the precision. Optional field.
len	Use *h* to truncate numeric value to 16 bits before conversion and in Tcl 8.4+, use *l* to insure value is 64 bits. Default is to use width of native machine word. Optional field.
char	Conversion type. See options below. Required field.

Possible values for flag are:

-	left-justified
+	always signed
0	zero pad
space	space pad
#	alt output (0 for oct, 0x for hex, include "." for fp, keep 0 for %g)

Possible values for char are:

d	signed decimal	**c**	int to char
u	unsigned decimal	**s**	string
i	signed decimal (#, 0x#, or 0#)	**f**	float (fixed)
o	unsigned octal	**e**	float (0e0)
x	unsigned hex	**E**	float (0E0)
X	unsigned HEX	**g**	auto float (f or e)
%	plain %	**G**	auto float (f or E)

regexp *?options? ?--? exp string ?matchVar? ?subMatchVar ...?*

Returns 1 if the regular expression *exp* matches part or all of *string*, 0 if not. If specified, *matchVar* will be set to the matching characters and the *subMatchVar*'s will be set to parenthesized subexpressions starting with the leftmost one. Unused *subMatchVar*'s will contain "**-1 -1**" if -indices was used or to an empty string otherwise. See Regular Expressions. Leave out vars if only matching is needed. To pre-compile use "set re {...}; regexp $re {}". Options are:

	-about	(Tcl 8.2.3+) Instead of matching*exp*, returns list with info on *exp*. First element is subexp count and second is a list of property names of *exp* attributes.
	-all	(Tcl 8.3+) Match *exp* as many times as possible in the string, Vars will contain info on last match.
	-expanded	(Tcl 8.2.3+) Use expanded regular expressions and ignore comments and white-space.
	-indicies	Instead of storing matching chars in *subMatchVar*, store start and ending indices of match in *string* .
	-inline	(Tcl 8.3+) Return list of data that would have been stored in *matchVar* and *subMatchVar*. Used with **-all**, each iteration will have match data and each subexpression concatenated to list.
	-line	(Tcl 8.2.3+) Enables newline-sensitive matching. Equivalent to using both **-linestop** and **-lineanchor** or **(?n)** embedded option.
	-lineanchor	(Tcl 8.2.3+) Changes behavior of "^" and "$" anchors so they match the start and end of a line, respectively. Same as **(?w)** embedded option.
	-linestop	(Tcl 8.2.3+) Changes behavior of "[^" bracket expressions and "." so that they stop at newlines. Same as **(?p)** embedded option.
	-nocase	Ignore case in matching.
	-start *index*	(Tcl 8.3+) Specifies char *index* offset to start matching *exp* at. With **-indicies**, the indices will be in terms of the absolute beginning. "^" will not match line start.
regsub *?options? ?--? exp string subSpec ?varName?*		Substitute first match of regular expression *exp* in *string* with *subSpec* and put in *varName* (default is to return matched portion in Tcl 8.4+) if specified, and return a count of replacements made. Subspec's "&" or "\0", are replaced with the matching string and "\#" where # is [1-9], replaces the #th matched *exp* in *string*. See Regular Expressions. Options are:
	-all	Substitute *exp* with *subSpec* as many times as possible in the string.
	-expanded	(Tcl 8.2.3+) Use expanded regular expressions and ignore comments and white-space.
	-line	(Tcl 8.2.3+) Enables newline-sensitive matching. Equivalent to using both **-linestop** and **-lineanchor** or **(?n)** embedded option.
	-lineanchor	(Tcl 8.2.3+) Changes behavior of "^" and "$" anchors so they match the start and end of a line, respectively. Same as **(?w)** embedded option.
	-linestop	(Tcl 8.2.3+) Changes behavior of "[^" bracket expressions and "." so that they stop at newlines. Same as **(?p)** embedded option.
	-nocase	Ignore case in matching.
	-start *index*	(Tcl 8.3+) Specifies char *index* offset to start matching *exp* at. "^" will not match line start.

scan *string format varName ?varName ...?*

Parse *string* using *format* conversions, store results in *varName*s, and return a count of conversions performed or -1 if none. *Format* is in the form of **%**[*][*argpos$*][*width*][*size*]*char*. White-space in the data is skipped except for **c** or **[ ]** set conversions. In Tcl 8.3+, will return a list if no variables are specifed. Fields are:

Field	Description
*	Indicates that the converted value is to be discarded instead of assigned to a variable.
argpos	Specifies arg to use for value with scan "{arg #}$". Argpos can be a variable by using ${var}$ or if quoting "${var}\$". Uses successive args for * specifiers. If any positional specifiers are used, then all conversions must use them. In Tcl 8.3.3+, if #**$** is used after **%** then #*varName* is used instead. Optional field.
width	Integer maximum field width. Optional field.
size	Field size modifier. In Tcl 8.4+, use *l* or *L* to insure value is stored as 64 bits. Default is to truncate to width of native machine word. Optional field.
char	Conversion type. See options below. Required field.

Numeric Conversion Types:				Character ConversionTypes:	
d	signed dec int	**c**	char to int	**s**	string (non-white-space)
o	octal	**e** or **f**	float (0e0 or 0E0)	**[***abc***]**, **[***a-c***]**	chars in given range
x	hex	**g**	auto float (f or e)	**[^***abc***]**, **[^***a-c***]**	chars not in given range
u	unsigned int (Tcl 8.1+)	space or tab	any amount of white-space (without %)	**n**	Store # chars scanned in *varName* (Tcl 8.2+)
i	int (Tcl 8.1+) dec,hex,oct				

string bytelength *string*

(Tcl 8.1.1+) Returns number of bytes for UTF-8 encoding of *string*.

string compare *?options? string1 string2*

Lexicographically (ASCII value) compares *string1* to *string2* and returns -1, 0, or 1 if *string1* is less than, equal to, or greater than *string2*, respectively. *Options* are:

- **-nocase** — (Tcl 8.1.1+) Ignore case
- **-length** *number* — (Tcl 8.1.1+) Only compare first *number* of characters.

string equal *?options? string1 string2*

(Tcl 8.1.1+) Compare *string1* to *string2* character by character and return 1 if they are identical, 0 if not. *Options* are:

- **-nocase** — Ignore case
- **-length** *number* — Only compare first *number* of characters.

string first *string1 string2 ?startIndex?*

Returns the index of the first char of the first occurance of the exact match of *string1* in *string2*, -1 if none. In Tcl 8.1.1+, *startIndex* specifies the offset of the first char to use in the search and can be **end** or **end-***number* .

string index *string index*

Returns the character at *index* in *string*. If *index* < 0 or > **end**, returns empty string. In Tcl 8.1.1+, *index* can be **end** or **end-***number* .

string is *class ?options? string*

(Tcl 8.1.1+) Returns 1 if *string* is a valid member of *class* (see Regular Expression Character Classes), 0 if not. *Options* are:

- **-strict** — An empty string will not match (default is it always will).
- **-failindex** *varName* — If not a member, the index in the string where *class* is no longer valid will be stored in *varName*.For **boolean**, **true**, and **false**, if 0 is returned, *varName* will also be set to 0. For **double**, **integer**, or **wide integer**, an under/overflow will return 0 and *varName* will be set to -1.

string last *string1 string2 ?lastIndex?*

Return the index of first char in the last occurance of the exact match of *string1* in *string2*, -1 if none. In Tcl 8.1.1+, *lastIndex* specifies the offset of the last char to use in the search and can be **end** or **end-***number*.

string length *string*	Returns the number of characters in *string* based on the encoding or bytes for binary data.
string map *?***-nocase***? charMapList string*	(Tcl 8.1.1+) Replaces characters in *string* based on and in the order of the key/value pairs in *charMapList*.*CharMapList* is a list of *key/value* pairs (can be multiple chars) as the elements. Case is ignored if **-nocase** is used.
string match *?***-nocase***? pattern string*	Returns 1 if *pattern* matches *string* using Pattern Globbing, 0 if not. In Tcl 8.1.1+, case will be ignored with **-nocase**.
string range *string first last*	Returns characters in *string* between indices *first* and *last* inclusive. If *first* < 0, then 0 is used. If *last* > **end**, then **end** is used. If *first* > *last*, then empty string is returned. *First* and *last* can be **end**. In Tcl 8.1.1+, *first* and *last* can also be **end-***number*.
string repeat *string count*	(Tcl 8.1.1+) Returns *string* repeated *count* times.
string replace *string first last ?newString?*	(Tcl 8.1.1+) Replaces characters in *string* between indices *first* and *last*, inclusive, with *newString* (default is to delete chars). If *first* < 0, then 0 is used. If *last* > **end**, then **end** is used. If *first* > *last*, *first* > string length, or *last* < 0, then *string* is returned unchanged.
string tolower *string ?first? ?last?*	Returns new string formed by converting all chars in *string* to lower case. In Tcl 8.1.1+, a subset of the string between indicies *first* and *last*, inclusive, can be converted. *First* and *last* can be **end** or **end-***number* .
string totitle *string ?first? ?last?*	(Tcl 8.1+) Returns new string formed by converting the first char in *string* to title case (or upper case if no title case equivalent) and the rest to lower case. If specified, a subset of the string between indicies *first* and *last*, inclusive, can be converted. *First* and *last* can be **end** or **end-***number*.
string toupper *string ?first? ?last?*	Returns new string formed by converting all chars in *string* to upper case. In Tcl 8.1.1+, a subset of the string between indicies *first* and *last*, inclusive, can be converted. *First* and *last* can be **end** or **end-***number* .
string trim *string ?chars?*	Returns 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?*	Same as **string trim** for leading characters only.
string trimright *string ?chars?*	Same as **string trim** for trailing characters only.
string wordend *string index*	Returns index in *string* of char just after last char in the word containing *index*.
string wordstart *string index*	Returns index in *string* of first char in the word containing *index*.
tcl_endOfWord *string start*	Returns the index of the first end-of-word location that occurs after a starting index *start* in the string *string* or -1 if none remain.
tcl_startOfNextWord *string start*	Returns the index of the first start-of-word location that occurs after a starting index *start* in the string *string* or -1 if none remain.
tcl_startOfPreviousWord *string start*	Returns the index of the first start-of-word location that occurs before a starting index *start* in the string *string* or -1 if none remain.
tcl_wordBreakAfter *string start*	Returns the index of the first word boundary after the starting index *start* in the string *string* or -1 if no more boundaries.
tcl_wordBreakBefore *string start*	Returns the index of the first word boundary before the starting index *start* in the string *string* or -1 if no more boundaries.

2.18 Variables

Command	Description
global *varName ?varName ...?*	(See Procedures)
incr *varName ?increment?*	Increment the integer value stored in *varName* by *increment* (default is 1). Max increment value is pow(2,32)
set *varName ?value?*	Store *value* in *varName* for current scope and namespace. Without value, returns the current value of *varName*. Can use namespace qualifiers in varName to specify a namespace or ::VarName for global variables. See Syntax for variable substitution forms.
set *arrayName(index) ?value?*	Same as above except for array element.
trace add *type name opList command*	(TCL 8.4+) Adds Tcl commands to be executed whenever certain operations are invoked. *Types* are:
command	Arrange for *command* to be executed whenever command *name* is modified based on *opList* (**rename** or **delete**). Args appended to *command* at execution are the oldCmdName, newCmdName, and *opList*. For **delete** newName is empty string. In Tcl 8.4.2+, the command name is fully qualified.
execution	Arrange for *command* to be executed whenever command *name* is executed based on *opList*. Available *opList* options are: **enter** (before execution start), **leave** (after execution completes), **enterstep** (before each command in *name* is executed), or **leavestep** (after each command in *name* is executed). For **enter** and **enterstep**, args appended to *command* at execution are the command-string (complete cmd being executed) and *opList*. For **leave** and **leavestep**, appended to *command* at execution are the command-string (complete cmd being executed), code (exec result code), result (exec result string), and *opList* .
variable	Arrange for *command* to be executed whenever variable *name* is accessed or modified based on *opList*. Available *opList* options are: **array** (via **array** cmd), **read** (variable is read), **write** (variable is written), or **unset** (variable is unset). Args appended to *command* when executed are name1 (scalar var name or array name), name2 (empty string or array index, if not whole array), and *opList*.
trace info *type name*	(TCL 8.4+) Returns list (where each element is a two element list of *opList* and *command* pairs) of trace operations currently set for command or variable *name*. Options for *type* are the same as **trace add**.
trace remove *type name opList command*	(TCL 8.4+) Removes trace on Tcl commands or variables to be executed as defined in **trace add** operation. Options for *type*, *name*, *opList*, and *command* are the same as **trace add** except *opList* can be a list of *opList* to use.
trace variable *varName ops command*	Same as **trace add variable** *varName ops command*, except *ops* is not a list and can be a string of: **a** for array, **r** for read, **w** for write, and/or **u** for unset.
trace vdelete *varName ops command*	Same as **trace remove variable** *varName ops command*, except *ops* is not a list and can be a string of: **a** for array, **r** for read, **w** for write, and/or **u** for unset.
trace vinfo *varName*	Same as **trace info variable** *varName*.
unset*?***-nocomplain***??--?* *name ?name ...?*	Removes the variables or arrays *name* from scope. If *name* is an array(index) in an array, only that element is removed. If its just an array name then the whole array is deleted. See Syntax for variable substitution forms. In Tcl 8.4+, **-nocomplain** suppresses any possible errors.
uplevel *?level? arg ?arg ...?*	Evaluates concatenation of *args* in the variable context indicated by*level* (default is 1). *Level* is an integer that gives the distance up the calling stack or with a prefix of "#", the absolute level number down the stack from global level #0. Returns result of evaluation. There is a performance impact if level is not specified.
upvar *?level? otherVar localVar ?otherVar localVar ...?*	Links *localVar* in local scope to *otherVar* in the variable context indicated by *level* (default is 1) so they share the same storage space. *LocalVar* must be scalar (Tcl 8.5+ will return an error for an array), but *otherVar* can be scalar, an array, or an array element. Level has the same definition as uplevel. The **unset** operation affects the linked to variable otherVar and not the upvared variable localVar. Traces on *otherVar* (except for entire arrays) will also work for *localVar*, but the variable returned will be *localVar*.
variable	(See Namespaces)

3 Tk Commands

3.1 Bindings and Events

Command	Description
bind *tag*	Returns a list of all sequences for which there exist bindings for window *tag*. See Tags for *tag* format options.
bind *tag sequence*	Returns the script bound to *sequence* for window *tag* or empty string if none. See Event SequencePatterns for *sequence* format options.
bind *tag sequence script*	Create a binding to evaluate *script* at global level by the same interpreter whenever event in *sequence* occurs within window *tag*. If *script* is prefixed with "+" (within braces if used), it is appended to the existing binding. If *script* is an empty string, the current binding is removed. See Event Generation and Substitutions for *script* % substitutions. The script can contain **continue** to terminate current script and **break** to terminate current script and skip remaining scripts. If an error occurs during the *script* execution, **bgerror** will be executed at the global level.
bindtags *window ?tagList?*	Change tags and tag order for *window* to contents of list *tagList*. If *tagList* is an empty list, the tags are set back to the default (window name, window class, toplevel window, and **all**). Without *tagList*, the current set of binding tags is returned.
event add *<<virtual>> sequence ?sequence ...?*	Define a virtual event by triggering virtual event *virtual* whenever any one of the *sequence*s occur. See Event Sequence Patterns for *sequence* format options.
event delete *<<virtual>> ?sequence ...?*	Deletes each *sequence* (or all without *sequence*) from the trigger list for virtual event*virtual* . Ignores *sequences* not associated with virtual event *virtual* . See Event SequencePatterns for *sequence* format options.
event generate *window event ?option value ...?*	Generates a window event in *window* as if it had came from the window system. See Event Sequence Patterns for *event* format options. See Event Generation and Substitutions for *options*. The **-when** options are:
-when now	process immediately (default without **-when**)
-when tail	place at end of event queue
-when head	place at beginning of event queue
-when mark	same as head but behind previous generated **-when mark** events
event info *?<<virtual>>?*	Returns a list where each element is a sequence that triggers virtual event *virtual*. Without *virtual*, returns a list of all defined virtual events.

Tags

Each window has an associated list of tags, and a binding applies to a particular window if its tag is among those specified for the window. The supported tag formats are: .a.b.c format (path name for window) or an arbitary string. When a window or widget is destroyed, its bindings are also deleted but not bindings to the tags associated with the window. The default binding tags behavior and order is:

Order	Tag	Applicability
1	*internal window name*	applies to just that window
2	*toplevel window name*	applies to top level and all its internal windows
3	*widget class name*	applies to all widgets in class
4	**all**	applies to all windows in application

When used for items within a canvas or text widget, bindings for items will be invoked before bindings for the window as a whole. The binding order is:

Order	Binding	Description
1	**all**	binding associated with **all** tag
2	*item tag*	one binding for each of the item's tags (in order)
3	*item id*	binding associated with item's id

Event Sequence Patterns

The *sequence* argument is a list of one or more event patterns with optional white space between the patterns. An *event* pattern may be one of the following forms:

Event Pattern	Description
ASCII char	a single ASCII character (except space or "<") that matches a **Keypress** event
<*modifier-modifier-type-detail*>	String with zero or more *modifier*s (see Modifiers below), an event *type* (see Event Types below), and a *detail* field (see Details below) identifying a particular button or keysym, separated by white space or dashes. Any field may be omitted as long as at least one of *type* and *detail* is present. Shortcuts for keyboard events: <Keypress-x>, <Key-x>, <x>, x. Shortcuts for mouse button events: <ButtonPress-1>, <Button-1>, <1>.
<<*name*>>	User-defined virtual event of name *name* . Modifiers may not be combined with a virtual event. Binding to a virtual event may be performed before the virtual event is defined. If the virtual event definition changes, all windows bound to that virtual event will respond immediately to the new definition. See Default Virtual Events for default events.

Modifiers:

Modifiers are used to modify button or key events. Button is the associated mouse button. Mod is the associated modifier key. **Meta** and **M** refer to whichever of the **M1** through **M5** modifiers is associated with the meta key(s) on the keyboard (keysyms Meta_R and **Meta_L**) or none if no match. **Double**,**Triple**,**Quadruple** refer to multiple mouse clicks within the time-out period or other repeating events. In Tk 8.5+ for MS Windows, the Extended modifier appears for events that are associated with the keys on the "extended keyboard." On a US keyboard, the extended keys include the Alt and Control keys at the right of the keyboard, the cursor keys in the cluster to the left of the numeric pad, the NumLock key, the Break key, the PrintScreen key, and the / and Enter keys in the numeric keypad.

Command (Mac)	**Button-1** or **B1** (left)	**Mod1** or **M1** (Num Lock)	**Meta** or **M**
Control	**Button-2** or **B2** (middle)	**Mod2** or **M2** (Alt)	**Double**
Shift	**Button-3** or **B3** (right)	**Mod3** or **M3** (Scroll Lock)	**Triple**
Lock	**Button-4** or **B4**	**Mod4** or **M4** (Extended, Tk 8.5+ MS Windows)	**Quadruple** (Tk 8.3+)
Alt	**Button-5** or **B5**	**Mod5** or **M5**	

Event Types:

Type	Description
Activate	Toplevel window of sub-window has been activated (Mac, Windows)
ButtonPress, **Button**	Button is pressed
ButtonRelease	Button is released
Circulate	Window stacking order has changed (not supported on MS Windows)
CirculateRequest	(Tk 8.4+) Generated when an application wants its windows raised/lowered. Window Manager use only.
Colormap	Color map has changed
Configure	Window size, position, border, or stacking order has changed
ConfigureRequest	(Tk 8.4+) Generated when an application wants its toplevel window moved or resized. Window Manager use only.
Create	(Tk 8.4+) Generated when a new window is created
Deactivate	Toplevel window of sub-window has been deactivated (Mac, Windows)
Destroy	Window has been destroyed (after destroy)
Enter	Mouse has entered window
Expose	Window has been exposed (needs to redrawn which is handled by TK)
FocusIn	Window has received keyboard focus

Type	Description
FocusOut	Window has lost keyboard focus
Gravity	Window has moved due to change in the size of parent window
KeyPress, **Key**	Key is pressed
KeyRelease	Key is released
Leave	Mouse is leaving window
Map	Window has been remapped (opened or restored)
MapRequest	(Tk 8.4+) Generated when an application wants its main window mapped to the screen. Window Manager use only.
Motion	Mouse is moving in window
MouseWheel	(Tk 8.0.4+) Mouse scroll wheel has moved
Property	Window property has changed or been deleted (X11 only)
Reparent	Window has changed parents
ResizeRequest	(Tk 8.4+) Generated when an application wants to have its main window resized. Window Manager use only.
Unmap	Window has been unmapped (iconified or forgotten by geometry manager)
Visibility	Window has changed visibility (For MS Windows, this is only for entire window)

Details:

Event Type	Detail	Result
ButtonPress, **ButtonRelease**	button number (1-5)	If a button number is specified, only an event on that particular button will match and type will default to **ButtonPress**, otherwise an event on any button will match.
KeyPress, **KeyRelease**	keysym	Keysyms are textual specifications for the keys on the keyboard. See Keysyms below. If specified, type will default to **Keypress**.

Keysyms:

Commonly used keysyms for the *detail* field are 0-9, A-Z, a-z, and those in the table below. Complete list is available in /usr/include/X11/keysymdef.

Alt_L	**comma**	**F9**	**KP_Decimal**	**Next**	**Scroll_Lock**
Alt_R	**Control_L**	**F10**	**KP_Divide**	**nobreakspace**	**Select**
ampersand	**Control_R**	**Find**	**KP_Enter**	**numbersign**	**semicolon**
App	**degree**	**greater**	**KP_Equal**	**Num_Lock**	**Shift_L**
asciicircum	**Delete**	**Help**	**KP_F1**	**parenleft**	**Shift_Lock**
asciitilde	**diaeresis**	**Home**	**KP_F2**	**parenright**	**Shift_R**
asterisk	**dollar**	**Hyper_L**	**KP_F2**	**Pause**	**slash**
at	**Down**	**Hyper_R**	**KP_F4**	**percent**	**space**
backslash	**End**	**hyphen**	**KP_Multiply**	**period**	**Super_L**
BackSpace	**equal**	**Insert**	**KP_Separator**	**periodcentered**	**Super_R**
bar	**Escape**	**KP_0**	**KP_Space**	**plus**	**Sys_Req**
Begin	**exclam**	**KP_1**	**KP_Subtract**	**plusminus**	**Tab**
braceleft	**Execute**	**KP_2**	**KP_Tab**	**Print**	**underscore**
braceright	**F1**	**KP_3**	**Left**	**Prior**	**Undo**
bracketleft	**F2**	**KP_4**	**less**	**question**	**Up**
bracketright	**F3**	**KP_5**	**Linefeed**	**quotedbl**	**Win_L**
Break	**F4**	**KP_6**	**Menu**	**quoteleft**	**Win_R**
Cancel	**F5**	**KP_7**	**Meta_L**	**quoteright**	
Caps_Lock	**F6**	**KP_8**	**Meta_R**	**Redo**	
Clear	**F7**	**KP_9**	**minus**	**Return**	
colon	**F8**	**KP_Add**	**Multi_key**	**Right**	

Default Virtual Events:

Tk Ver	Virtual Event	Event Pattern (except text widget)	Text Widget	Unix	Windows	Mac (Aqua)
All	**<<Clear>>**					<Clear>
All	**<<Copy>>**	**<Control-c>**	**<Meta-w>**	**<F16>**	**<Control-Insert>**	**<F3>**
All	**<<Cut>>**	**<Control-x>**	**<Control-w>**	**<F20>**	**<Shift-Delete>**	**<F2>**
8.4+	**<<Modified>>**					
All	**<<Paste>>**	**<Control-v>**	**<Control-y>**	**<F18>**	**<Shift-Insert>**	**<F4>**
8.0.3+	**<<PasteSelection>>**	**<ButtonRelease-2>**				
All	**<<PrevWindow>>**	**<Shift-Tab>**				
8.4+	<<Redo>>			<Control-Z>	<Control-y>	<Control-y>
8.4+	**<<Selection>>**					
8.4+	<<Undo>>	<Control-z>	<Control-underscore>			
8.5+	<<TraverseIn>>					
8.5+	<<TraverseOut>>					

Binding Matches

Trigger	Action
If several bindings to match a given X event but have different tags	Each binding is executed. The default order is: binding for the widget, class binding, binding for its toplevel, and the **all** binding.
If several bindings match a given X event and they have the same *tag*	The most specific binding is chosen and its script is evaluated. See Order of Tests below for criteria to determine most specific binding.
If the matching sequences contain more than one event	Tests 3 to 5 in Order of Tests below are applied in order from the most recent event to the least recent event in the sequences. If these tests fail to determine a winner, then the most recently registered sequence is the winner.
If there are two or more virtual events triggered by the same sequence, and those virtual events are bound to the same tag	Only one of the virtual events will be triggered and it will be picked at random
A given X event does not match any of the existing bindings	The event is ignored. An unbound event is not considered to be an error.
When a *sequence* specified in a **bind** command contains more than one event pattern	Its script is executed whenever the recent events (leading up to and including the current event) match the given sequence. (ex. **Triple** will also match **Double**).

Order of Tests

Sequence	Test
1	pattern that specifies specific button or key
2	longer sequence of events matched
3	more matching modifiers
4	physical pattern not associated with a virtual event
5	undefined match for two or more virtual events

Event Generation and Substitutions

Binding scripts can contains % substitution codes to insert details about the event. When executed, a new script is generated which replaces the substitution codes with an properly formatted list containing the specified information from the current event. Invalid substitutions are undefined.

Event Generate Option	Bind Code	Description	Valid Events
	%%	Percent sign	all events
	%A	Substitute ASCII (pre Tk 8.2) or ISO Latin 1 (Tk 8.2+) char for event or empty string {} if none	**KeyPress**, **KeyRelease**
-above *window*	%a	*above* field for event where *window* is a path name or integer window id	**Configure**
-borderwidth *size*	%B	*border_width* field for event where *size* is distance	**Configure**, **ConfigureRequest**, **Create**
-button *number*	%b	Button *number* for event (*detail* field)	**ButtonPress**, **ButtonRelease**
-count *number*	%c	*count* field for event	**Expose**, **Map**
-data *string*	%d	Specifies user data field.	Only valid for virtual events.
-delta *number*	%D	(Tk 8.4+) reports the *delta* value where sign represents direction.	**MouseWheel**
-detail *detail*	%d	*detail* field for event. See below for *detail* enums.	**Enter**, **Leave**, **FocusIn**, **FocusOut**

-focus *boolean*	%f	*focus* field for event	**Enter**, **Leave**
-height *size*	%h	*height* field for event	**Configure**, **ConfigureRequest, Expose**
	%i	(Tk 8.4+) *window* field for event as a hex number	**CreateNotify**
-keysym *name*	%K	*keysym* for event as a text string	**KeyPress**, **KeyRelease**
-keycode *number*	%k	*keycode* field for event	**KeyPress**, **KeyRelease**
	%N	*keysym* for event as a decimal number	**KeyPress**, **KeyRelease**
-mode *notify*	%m	*mode* field for event. See below for *notify* enums.	**Enter**, **Leave**, **FocusIn**, **FocusOut**
-override *boolean*	%o	*override_redirect* field for event	**Map**, **Reparent**, **Configure**,**ConfigureRequest**
	%P	(Tk 8.4+) substitute the atom name for the property being changed	**PropertyNotify**
-place *where*	%p	*place* field for event. See below for *where* enums.	**Circulate**, **CirculateRequest**
-root *window*	%R	root window path name or ID for event	**KeyPress**, **KeyRelease**,**ButtonPress, ButtonRelease**, **Enter**, **Leave**, **Motion**
-rootx *coord*	%X	*x_root* field for event.	**KeyPress**, **KeyRelease**,**ButtonPress**,**ButtonRelease**, **Enter**, **Leave**, **Motion**
-rooty *coord*	%Y	*y_root* field for event.	**KeyPress**, **KeyRelease**,**ButtonPress**,**ButtonRelease**, **Enter**, **Leave**, **Motion**
-sendevent *boolean*	%E	*send_event* field for event. True for event generate, false for system generated.	all events
-serial *number*	%#	*serial* number for event	all events
-state *state*	%s	*state* field for event. See below for enums.	all events
-subwindow *window*	%S	*subwindow* ID for event	**KeyPress**, **KeyRelease**,**ButtonPress**,**ButtonRelease**, **Enter**, **Leave**, **Motion**
-time *integer*	%t	*time* field for event	**KeyPress**, **KeyRelease**,**ButtonPress**,**ButtonRelease**, **Enter**, **Leave**, **Motion**, **Property**
	%T	*type* field for event	all events
-warp *boolean*		(Tk 8.3+) Whether screen pointer should warp	**KeyPress**, **KeyRelease**,**ButtonPress**,**ButtonRelease**, **Motion**
	%v	value_mask field for event	**Configure**, **ConfigureRequest**
-width *number*	%w	*width* field for event	**Configure**, **ConfigureRequest**
	%W	path name of the window/widget to which the event was reported	all events
-x *coord*	%x	x field (relative) for event	**KeyPress**, **KeyRelease**,**ButtonPress**,**ButtonRelease**, **Enter**, **Leave**, **Motion**, **Expose**,**Configure**, **ConfigureRequest**, **Gravity**, **Reparent**
-y *coord*	%y	y field (relative) for event	**KeyPress**, **KeyRelease**,**ButtonPress**,**ButtonRelease**, **Enter**, **Leave**, **Motion**,**Expose**,**Configure**, **ConfigureRequest**, **Gravity**, **Reparent**

For some of the above substitutions, the possible replacement strings are:

Code	Event	Replacement String
%d	**Enter**, **Leave**,**FocusIn, FocusOut**	**NotifyAncestor**, **NotifyDetailNone**,**NotifyInferior**,**NotifyNonlinear**, **NotifyNonlinearVirtual**, **NotifyPointer**,**NotifyPointerRoot**, or **NotifyVirtual**
%d	**ConfigureRequest**	**Above**, **Below**, **BottomIf**,**Opposite**, **None**, or **TopIf**
%m	**Enter**, **FocusIn**,**FocusOut**, **Leave**	**NotifyNormal**, **NotifyGrab**,**NotifyUngrab**, or **NotifyWhileGrabbed**
%p	**Circulate**,**CirculateRequest**	**PlaceOnTop** or **PlaceOnBottom**
%s	**ButtonPress**, **ButtonRelease**,**Enter**, **KeyPress**, **KeyRelease**, **Leave**, **Motion**	decimal integer
%s	**Visibility**	**VisibilityUnobscured**, **VisibilityPartiallyObscured**, or **VisibilityFullyObscured**

3.2 Button Widget

Command	Description
button *pathName ?options?*	Creates a button *pathName* with *options* and returns the new widget's path name. When invoked, *pathName* must not exist, but *pathName*'s parent should. A button widget can display text, a bitmap, or an image. Selecting a button will cause the associated command to be evaluated. Multiple fonts within a button text field are not supported.

Button Widget Options

Standard

See Common Options and Resources in Options and Resources for full details.

-activebackground	**-font**	**-relief**
-activeforeground	**-foreground**	**-repeatdelay** (Tk 8.4+)
-anchor	**-highlightbackground**	**-repeatinterval** (Tk 8.4+)
-background	**-highlightcolor**	**-takefocus**
-bitmap	**-highlightthickness**	**-text**
-borderwidth	**-image**	**-textvariable**
-compound (8.4+)	**-justify**	**-underline**
-cursor	**-padx**	**-wraplength**
-disabledforeground	**-pady**	

Button Widget Specific

See Coordinates in General Tk Widget Information for screen unit options.

Configure Option	Resource Name	Resource Class	Description
-command *script*	**command**	**Command**	Tcl command to associate with the button. *Script* is invoked when mouse button 1 is released over the button window.
-default *state*	**default**	**Default**	Sets platform specific appearance state of default ring. Options are: **active** (default button), **normal** (non-default button), or **disabled** (non-default button without leaving space for default ring).
-height *height*	**height**	**Height**	Height of button in screen units for bitmaps/images and in lines for text. Default is to auto size.
-overrelief *type*	**overRelief**	**OverRelief**	(Tk 8.4+) Alternative relief for when mouse cursor is over button. Not used when set to empty string (default). Options are: **flat**, **raised**, and **sunken**.
-state *state*	**state**	**State**	State of button. Options are: **active** (mouse pointer over button, use **activeforeground** and **activebackground**),**disabled** (button is insensitive, use **disabledforeground** and **background**), or **normal** (use **foreground** and **background**).
-width *width*	**width**	**Width**	Width of button in screen units for bitmaps/images and in characters for text. Default is to auto size.

Button Widget Commands

Command	Description
pathName **cget** *option*	Returns the current value of the configuration *option* for button *pathName*. See Button Widget Options above for *options*.
pathName **configure***?option? ?value? ?option value ...?*	Change the configuration *option* for the button *pathName* to *value*. Without *value*, a list describing the available options is returned. Without *option*, a list describing all of the available options for *pathName* is returned. For multiple options an empty string is returned. See Button Widget Options above for *options*.
pathName **flash**	Flash checkbutton by toggling between active and normal colors several times. Button is left is initial state of **active** or **normal**. Ignored if button is disabled.
pathName **invoke**	Toggle the selection state of the checkbutton and invoke the Tcl command specified with **-command**, if any. Returns value of Tcl command or empty string if no **-command** . Ignored if button is disabled.

Default Button Widget Bindings

Active or normal button default bindings:

Event	Description
<Enter>	When mouse passes over button, **relief** changes to **sunken** (Unix and MS Windows only) and state becomes **active**.
<Leave>	When mouse leaves the button, **relief** changes to **rasied** (Unix and MS Windows only) and state becomes **normal**.
<Button-1>	When button 1 is pressed, **relief** changes to **sunken** (Unix and MS Windows only) and state becomes**active** (Windows and Mac only)**.**
<ButtonRelease-1>	When button 1 is released, **relief** changes to **rasied** (Unix and MS Windows only) and state becomes**normal** (Windows and Mac only). If still over button, **-command** *script* is invoked.
<space>	If button has focus, **relief** changes to**sunken**, state becomes **active**, and **command** *script* is invoked.

3.3 Canvas Widget

Command	Description
canvas *pathName ?options?*	Creates a canvas widget *pathName* with *options* and returns the new widget's path name. When invoked, *pathName* must not exist, but *pathName*'s parent should. A canvas window can be used to show structured graphics. Widgets created within the canvas are referred to as items and are displayed in the order they are listed in the widget except for window items. Items are ordered from lowest (first) to highest (latest) in the display list such that later items can obscure earlier items. New canvases are not given any default binding behavior.

Canvas Options

Standard

See Common Options and Resources in Options and Resources for full details.

-background	**-insertbackground**	**-selectbackground**
-borderwidth	**-insertborderwidth**	**-selectborderwidth**
-cursor	**-insertofftime**	**-selectforeground**
-highlightbackground	**-insertontime**	**-takefocus**
-highlightcolor	**-insertwidth**	**-xscrollcommand**
-highlightthickness	**-relief**	**-yscrollcommand**

Canvas Specific

See Coordinates in Options and Resources for screen unit options.

Configure Option	Resource Name	Resource Class	Description
-closeenough *float*	**closeEnough**	**CloseEnough**	Value indicating how close the mouse cursor must be to an item before it is considered to be "inside" the item. Default is 1.0.
-confine *boolean*	**confine**	**Confine**	Specifies whether to confine the canvas view to the scroll region (default) or not.
-height *height*	**height**	**Height**	Height of canvas in screen units.
-scrollregion *left top right bottom*	**scrollRegion**	**ScrollRegion**	List of four coordinates describing the left, top, right, and bottom of a rectangular scrolling region.
-state *state*	**state**	**State**	(Tk 8.3+) State of canvas. Options are: **active** (when mouse pointer is over widget), **disabled**, or **normal**.
-width *width*	**width**	**Width**	Width of button in screen units.
-xscrollincrement *distance*	**xScrollIncrement**	**ScrollIncrement**	Specifies the increment for horizontal scrolling in screen units. If *distance* <=0, scrolling is unconstrained.
-yscrollincrement *distance*			Specifies the increment for vertical scrolling in screen units. If *distance* <=0, scrolling is unconstrained.

Item IDs and Tags

Each item in a canvas widget has an unique id and one or more tags. The symbol *tagOrId* is used to indicate that an argument specifies either an id that selects a single item or a tag that selects zero or more items. It may contain a logical expressions of tags by using operators: '**&&**', '||', '^' '**!**', and parenthezised subexpressions. When an item is destroyed, bindings to IDs are also deleted, but not bindings to tags.

ID/Tag	Description
unique id	Unique numeric identifier of item within a canvas. Items can only have one id.
tag	String of characters that other than a numeric value used to group items. Items can have multiple tags.
all	Tag associated with all items in a canvas widget.
current	Tag set automatically by Tk to the current item (topmost item) under the mouse pointer, if any.

Indicies or Character Positions:

Some canvas commands support the use of an index to locate the position of characters (text) or coordinates (line and polygon) within the canvas starting from 0. The indicies for lines and polygons are always even. The following are the valid forms of specifying an *index* :

Index form	Supported by	Description
number	text, line, polygon	A decimal number giving the position of the desired character within the text item where 0 = first character. Odd indicies for lines and polygons are decremented by 1. If *number* < 0, the 0 is used, if *number* > length of text list, then **end** is used. For polygons, if *number* < 0 or > length, then length is added or subtracted until *number* is in range.
end	text, line, polygon	Character or coordinate just after last one in item.
insert	text	Character just after the insertion cursor.
sel.first	text	First selected character in item.
sel.last	text	Last selected character in item.
@x,y	text, line, polygon	Character or coordinate at the point given by *x* and *y* using canvas coordinate system. If *x* or *y* are outside the item coordinates, they are set to the first or last character in line closest to given point.

Canvas Commands

The following are the valid command operations that can be invoked on the canvas widget *pathName* created by the **canvas** command. Widgets created within the canvas are referred to as items.

Command	Description
pathName **addtag** *tag searchSpec ?arg ...?*	For each item that matches *searchSpec* and *arg* in canvas *pathName*, add *tag* to the list of tags associated with that item. *SearchSpec* and *arg* options are:
above *tagOrId*	Selects the last (topmost) item in display list, just after (above) the one given by *tagOrId* in the display list.
all	Selects all the items in the canvas.
below *tagOrId*	Selects the first (lowest) item in display list, just before (below) the one given by *tagOrId* in the display list.
closest *x y ?halo? ?start?*	Select the last (topmost) item in display list, closest to **@***x,y*. If specified, it must be below *start* in the display list. Any item closer than *halo* to the point is considered to overlap it.
enclosed *x1 y1 x2 y2*	Selects all the items completely enclosed within rectangular region **@***x1,y1* and **@***x2,y2* where *x1* < *x2* and *y1* < *y2*.
overlapping *x1 y1 x2 y2*	Selects all the items that overlap or are enclosed within rectangular region **@***x1,y1* and **@***x2 ,y2* where *x1* < *x2* and *y1y2*. <
withtag *tagOrId*	Selects all the items given by *tagOrId*.
pathName **bbox***tagOrId ?tagOrId ...?*	Returns a list of four elements *x1 y1 x2 y2*, giving an approximate bounding box (rectangular region **@***x1,y1* and **@***x2,y2*) for all items named by the *tagOrId* args. If no *tagOrId* matches or items have empty bounding boxes, returns empty string.

pathName **bind***tagOrId ?sequence? ?command?*	Create a binding to evaluate *command* whenever event in *sequence* occurs within the items named by *tagOrId* . See **bind** command for options. Only mouse, keyboard, and virtual events can be used.
pathName **canvasx***screenx ?gridspacing?*	Returns the canvas x-coordinate that is displayed at window x-coordinate *screenx* rounding to nearest multiple of *gridspacing* units, if specified.
pathName **canvasy***screeny ?gridspacing?*	Returns the canvas y-coordinate that is displayed at window y-coordinate *screeny* rounding to nearest multiple of *gridspacing* units, if specified.
pathName **cget** *option*	Returns the current value of the configuration *option*. See Standard Options and Canvas Specific Options above for *options*.
pathName **configure** *?option? ?value? ?option value ...?*	Change the configuration *option* to *value*. Without *value*, a list describing *option* is returned. Without *option*, a list of all available options for *pathName* is returned. For multiple options an empty string is returned. See Canvas Options above for *options*.
pathName **coords***tagOrId ?x0 y0 ...? ?coordList?*	Change coordinates for first item given by *tagOrId* to specifed coordinates or *coordList* (Tcl 8.3+). Without coordinates, returns a list whose elements are coordinates of item *tagOrId*.
pathName **create***type ?x y ...? ?coordList? ?option value ...?*	Create a new item in *pathName* of type*type* (See Canvas Item Types below) at specified coordinates or *coordList* (Tk 8.3+) with *option*s. Returns id of new item.
pathName **dchars***tagOrId first ?last?*	For items given by *tagOrId*, delete the characters (text) or coordinates (line or polygon) in the range given by first and last (defaults to first), inclusive. See Indicies or Char Positions above.
pathName **delete***?tagOrId ...?*	Delete each of the items given by each *tagOrId* .
pathName **dtag***tagOrId ?tagToDelete?*	Delete the tag given by *tagToDelete* (default is *tagOrId*) from the list of associated tags for each item given by *tagOrId*.
pathName **find***searchSpec ?arg ...?*	Returns a list of items in stacking order that satisfy the specification *searchSpec*. See **addtag** for *searchSpec* options.
pathName **focus** *?tagOrId?*	Set the keyboard focus to the first item (lowest) given by*tagOrId* that supports the insertion cursor (text). If *tagOrId* is empty string, the focus item is unset. Without *tagOrId* , returns current item with focus or empty string if none.
pathName **gettags** *tagOrId*	Return a list of the tags associated with the first item (lowest) given by *tagOrId* or empty string if none.
pathName **icursor** *tagOrId index*	Set the insertion cursor for the item(s) given by *tagOrId* that support the insertion cursor (text) to just before the character at position *index*. See Indicies or Char Positions above. Does not effect keyboard focus.
pathName **index***tagOrId index*	Returns a decimal string giving the numerical index of the first item (lowest) within *tagOrId* corresponding to the character (text) or coordinate (line or polygon) at position *index*. See Indicies or Char Positions above.
pathName **insert***tagOrId beforeThis string*	For items given by *tagOrId* that support text or coordinate insertion, insert *string* just before character (text) or coordinate (line or polygon) at position *beforeThis* . For lines or polygons, *string* must be a valid coordinate sequence. See Indicies or Char Positions above.
pathName **itemcget***tagOrId option*	Returns the current value of the configuration *option* for the first item (lowest) given by *tagOrId*. See Canvas Options above for *options*.
pathName **itemconfigure***tagOrId ?option? ?value? ?option value ...?*	Change the configuration *option* for item *tagOrId* to *value*. Without *value*, a list describing *option* is returned. Without *option*, a list of all available options for the first item (lowest) given by *tagOrId* is returned. For multiple options an empty list is returned. See Canvas Options above for *options*.
pathName **lower***tagOrId ?belowThis?*	Move items given by *tagOrId* to a new position on the display list just before the first (lowest) item given by *belowThis* . If *tagOrId* refers to more than one item, then all items are moved, but their relative order remains the same.
pathName **move***tagOrId xAmount yAmount*	Move each of the items given by *tagOrId* in the canvas coordinate space by adding *xAmount* and *yAmount* to each items x and y coordinates, respectively.

pathName **postscript***?option value ...?*	Generate a Encapsulated Postscript representation for part or all of the canvas. Options are:
-channel *channelID*	Specifies the *channelID* to write the Postscript code to.
-colormap *varName*	Specifies a color mapping array *varName* where each element is a color name and the value is Postscript code to set a particular color value. If not set or for unspecified colors, Tk uses the RGB intensities.
-colormode *mode*	Specifies how to output color information where *mode* is: **color**, **gray** (grayscale), or **mono** (black or white).
-file *fileName*	Specifies the file to write the Postscript code to. Not vaild for safe interpreters. If not specified, the Postscript is returned as the result of the command.
-fontmap *varName*	Specifies a font mapping array *varName* where each element is a list of two elements consisting of the name and point size of a postscript font and the value is Postscript code to set a particular font. If not set or for unspecified fonts, Tk attempts to guess. See Fonts for font options.
-height *size*	Specifies the height (default is canvas window height) of the area of the canvas to print. See Coordinates in Options and Resources for screen unit options.
-pageanchor *anchor*	Specifies which point (default is **center**) of the printed area should be appear over the positioning point on the postscript page.
-pageheight *size*	Specifies that the Postscript should be scaled in both x and y directions so that the printed area is *size* high (default is height on screen) on the Postscript page. See Coordinates in Options and Resources for screen unit options.
-pagewidth *size*	Specifies that the Postscript should be scaled in both x and y directions so that the printed area is *size* wide (default is width on screen) on the Postscript page. See Coordinates in Options and Resources for screen unit options. Supercedes **-pageheight** option.
-pagex *position*	Set the x-coordinate of the positioning point on the postscript page to *position* (default is **center**). See Coordinates in Options and Resources for screen unit options.
-pagey *position*	Set the y-coordinate of the positioning point on the postscript page to *position* (default is **center**). See Coordinates in Options and Resources for screen unit options.
-rotate *boolean*	If true, the printed area is to be rotated 90 degrees for landscape orientation (default is false for portrait).
-width *size*	Specifies the width (default is canvas window width) of the area of the canvas to print. See Coordinates in Options and Resources for screen unit options.
-x *position*	Specifies the x-coordinate of the left edge of canvas area (default is left edge of canvas window) to print in canvas coordinates.
-y *position*	Specifies the y-coordinate of the top edge of canvas area (default is top edge of canvas window) to print in canvas coordinates.
pathName **raise***tagOrId ?aboveThis?*	Move all items given by *tagOrId* to a new position on the display list just after the last (topmost) item given by *aboveThis*. If *tagOrId* refers to more than one item, then all items are moved, but their relative order remains the same.
pathName **scale***tagOrId xOrigin yOrigin xScale yScale*	Rescale all items given by *tagOrId* in canvas coordinate space to change the distance from **@***xOrigin*,*yOrigin* by a scale factor of *xScale*,*yScale* (1.0 = no change) respectively.
pathName **scan***option args*	Implements scanning on canvas widgets. *Options* are:
mark *x y*	Records *x* and *y* and the current view in canvas. Typically associated with mouse button press in widget at coordinates *x*,*y*.
dragto *x y ?gain?*	Adjusts the view in Tk 8.3+ by *gain* (default is 10 in all Tk versions) times the difference between the coordinates *x*,*y* and the last **mark** *x*,*y* coordinates. Used with mouse motion events to produce high speed dragging effect.
pathName **select***option ?tagOrId arg?*	Manipulates the selection based on the specified *option*. Where specified, the first item given by *tagOrId* that supports indexing and selection (text) is used. See Indicies or Char Positions above for *index* options.

adjust *tagOrId index*	Adjust the end of the selection nearest to the character given by *index* in *tagOrId* to include up to *index* and set the other end to be the anchor point. Works the same as **select to** if selection is not in *tagOrId*.
clear	Clear the selection if it is in the widget.
from *tagOrId index*	Set the selection anchor point to be just before the character given by *index* in the item given by *tagOrId*.
item	Return id of the selected item or an empty string if there is none.
to *tagOrId index*	Set the selection to the characters in *tagOrId* from position *index* to the anchor point (included only if *index* > anchor point) in *tagOrId*. If the anchor point is not in *tagOrId*, *index* is used.
pathName **type***tagOrId*	Returns the type (see Canvas Item Types below) of the first item (lowest) given by *tagOrId* or empty string if none.
pathName **xview** ?*option args*?	Query or change the horizontal canvas widget view. Without any *options*, a two element list is returned specifying the start and end of the visible fraction (from 0 to 1) of the horizontal span of the widget between the left and right edges of the window. Vaild *options* and *args* are:
moveto *fraction*	Adjust the view in the window so that *fraction* (from 0 to 1) of the total width of the widget is off-screen to the left.
scroll *number* **pages**	Shift the view left or right in units of nine-tenths the window's width. If *number* < 0, information farther to the left becomes visible, otherwise information farther to the right becomes visible.
scroll *number* **units**	Shift the view left or right by *number* units. If *number* > 0, units is same as **xScrollIncrement** option, otherwise units is one-tenth of window's width.
pathName **yview**?*option args*?	Query or change the vertical canvas widget view. Without any *options*, a two element list is returned specifying the start and end of the visible fraction (from 0 to 1) of the vertical span of the widget between the top and bottom edges of the window. Vaild *options* and *args* are:
moveto *fraction*	Adjust the view in the window so that *fraction* (from 0 to 1) of the total height of the widget is off-screen to the top.
scroll *number* **pages**	Shift the view up or down in units of nine-tenths the window's height. If *number* < 0, then higher information becomes visible, otherwise lower information becomes visible.
scroll *number* **units**	Shift the view up or down by *number* units. If *number* > 0, units is same as **yScrollIncrement** option, otherwise units is one-tenth of window's height.

Canvas Item Standard Options

<table>
<tr><th>Normal State</th><th>Active State (Tk 8.3+)</th><th>Disabled State (Tk 8.3+)</th><th>Description</th></tr>
<tr><td>-dash pattern</td><td>-activedash pattern</td><td>-disableddash pattern</td><td>(Tk 8.3+) Specifies dash pattern for item. Where pattern is:
<table><tr><td>list of integers</td><td>Each element represents the number of pixels of a line segment. Only the odd segments are drawn using the "outline" color. The other segments are drawn transparant.</td></tr><tr><td>list containing [.,-_]</td><td>Character list containing only 5 possible characters. .={2 4}; ,={4 4}; -={6 4}; _={8 4} and space can be used to enlarge the space between other line elements, and can not occur as the first position in the string.</td></tr></table></td></tr>
<tr><td>-dashoffset offset</td><td></td><td></td><td>(Tk 8.3+) The starting offset in pixels into the pattern provided by the -dash option. See Coordinates in Options and Resources for screen unit options.</td></tr>
<tr><td>-fill color</td><td>-activefill color</td><td>-disabledfill color</td><td>Specifies the color to be used to fill an item’s area. See Colors in Options and Resources for color options.</td></tr>
<tr><td>-outline color</td><td>-activeoutline color</td><td>-disabledoutline color</td><td>Specifies the color to be used to draw the outline of an item. If set to an empty string, no outline is used. See Colors in Options and Resources for color options.</td></tr>
<tr><td>-offset offset</td><td></td><td></td><td>(Tk 8.3+) Specifies the offset of stipples. The offset value can be of the form x,y (origin is the canvas origin or with a "#" prefix it is the origin of the current toplevel window) or side (which can be: n,ne,e,se,s,sw,w,nw, or center). For the line and polygon items, adding an index argument connects the stipple origin to one of the coordinate points of the line/polygon.</td></tr>
<tr><td>-outlinestipple bitmap</td><td>-activeoutlinestipple bitmap</td><td>-disabledoutlinestipple bitmap</td><td>Specifies the stipple patterns to be used to draw the outline of the item. If set to an empty string, a solid outline is used. See Default Bitmaps in Options and Resources for bitmap options. Used with -outline.</td></tr>
<tr><td>-stipple bitmap</td><td>-activestipple bitmap</td><td>-disabledstipple bitmap</td><td>Specifies the stipple patterns to be used to be used to fill an item’s area. If set to an empty string, a solid fill is used. See Default Bitmaps in Options and Resources for bitmap options. Used with -fill.</td></tr>
<tr><td>-state state</td><td></td><td></td><td>Set to override canvas widget’s global state for item. State options are: normal, disabled, or hidden.</td></tr>
<tr><td>-tags tagList</td><td></td><td></td><td>Specifies a set of tags to apply to item. TagList is a list of tag names, which replace any existing tags for the item.</td></tr>
<tr><td>-width outlineWidth</td><td>-activewidth outlineWidth</td><td>-disabledwidth outlineWidth</td><td>Specifies the width of the outline (default is 1.0) to be drawn around the item. See Coordinates in Options and Resources for screen unit options. Used with -outline .</td></tr>
</table>

Canvas Item Commands

See Canvas Item Standard Options above for item standard options below.

Command	Description

pathName **create arc** *?x1 y1 x2 y2? ?coordList? ?option value ...?*

Display an arc-shaped region (oval delimited by two angles specified by **-start** and **-extent** options). Args *x1*,*y1* and *x2*,*y2* or *coordList* give the coordinates of two diagonally opposite corners of a rectangular region enclosing the oval that defines the arc. *Options* are:

-dash	**-activedash**	**-disableddash**	**-dashoffset**
-fill	**-activefill**	**-disabledfill**	
-outline	**-activeoutline**	**-disabledoutline**	
-outlinestipple	**-activeoutlinestipple**	**-disabledoutlinestipple**	
-stipple	**-activestipple**	**-disabledstipple**	**-offset**
-width	**-activewidth**	**-disabledwidth**	
-state			
-tags			

Arc Specific Options:

-extent *degrees*	Size of the angular range occupied by arc in degrees (range = Â±360, if outside range then *degrees* modulo 360) counter-clockwise from **-start** angle.
-start *degrees*	Starting angle (Â±) measured from 3-o'clock position.
-style *type*	Arc is drawn as either *type* **pieslice** (default) where the enclosed region is a section of the perimeter and two lines from the center to the perimeter endpoints; **chord** where the enclosed region is a section of the perimeter and a line connecting the perimeter endpoints; or **arc** where the enclosed region is just a section of the perimeter.

pathName **create bitmap** *?x y? ?coordList? ?option value ...?*

Display a bitmap at positioning point coordinates of *x*,*y* or *coordList*. *Options* are:

-state
-tags

Bitmap Specific Options:

-anchor *anchorPos*	Specifies how to position bitmap relative to item positioning point where *anchorPos* is **n**, **ne**, **e**, **se**, **s**,**sw**,**w** ,**nw**, or **center** (default).
-background *color*	Specifies color to use for background in normal state. If not specified or set to an empty string, then background is transparent. See Colors in Options and Resources for *color* options.
-activebackground*color*	(Tk 8.4+) Specifies color to use for background in active state.
-disabledbackground*color*	(Tk 8.4+) Specifies color to use for background in disabled state.
-bitmap *bitmap*	Specifies the bitmap to display in the normal state. See Default Bitmaps in Options and Resources for *bitmap* options.
-activebitmap *bitmap*	(Tk 8.4+) Specifies the bitmap to display in the active state.
-disabledbitmap *bitmap*	(Tk 8.4+) Specifies the bitmap to display in the disabled state.
-foreground *color*	Specifies color to use (default is black) for foreground in normal state. See Colors in Options and Resources for *color* options.
-activeforeground*color*	(Tk 8.4+) Specifies color to use for foreground in active state.
-disabledforeground*color*	(Tk 8.4+) Specifies color to use for foreground in disabled state.

pathName **create image** *?x y? ?coordList? ?option value ...?*

Display an image at positioning point coordinates of *x*,*y* or *coordList* .*Options* are:

-state
-tags

Image Specific Options:

-anchor *anchorPos*	Specifies how to position the image relative to the item positioning point where*anchorPos* is **n**, **ne**, **e**, **se**, **s**,**sw** ,**w**,**nw**, or **center** (default).
-image *image*	Specifies the image to display in the normal state.
-activeimage*image*	(Tk 8.4+) Specifies the image to display in the active state.
-disabledimage *image*	(Tk 8.4+) Specifies the image to display in the disabled state.

pathName **create line** *?x1 y1 ... xN yN? ?coordList? ?option value ...?*

Display one or more connected line segments or curves. Args *x1*,*y1* through *xN*,*yN* or *coordList* give the coordinates for a series of two or more points that describe a series of connected line segments. *Options* are:

-dash	**-activedash**	**-disableddash**	**-dashoffset**
-fill	**-activefill**	**-disabledfill**	
-stipple	**-activestipple**	**-disabledstipple**	
-width	**-activewidth**	**-disabledwidth**	
-state			
-tags			

Line Specific Options:

-arrow *where*	Specifies whether an arrowhead should be drawn at line endpoints. Options are: **none** (default option for no arrowheads), **first** (for an arrowhead at the first point of the line), **last** (for an arrowhead at the last point of the line), or **both** (for arrowheads at both ends).
-arrowshape *shape*	Specifies arrowhead shape where *shape* is a three element list (neck length to tip, trailing point length to tip, width from line to trailing point). Default is a reasonable shape. See Coordinates in Options and Resources for screen unit options.
-capstyle *style*	Specifies caps to be drawn at endpoints of the line. *Style* options are: **butt** (default), **projecting**, or **round**. Superceded by **-arrow**.
-joinstyle *style*	Specifies joints to be drawn at line verticies. *Style* options are: **bevel** ,**miter**, or **round**.
-smooth *smoothMethod*	Set to true or **bezier**, smoothing is used to draw the line as a curve. The line is rendered as a set of parabolic splines: one spline is drawn for the first and second line segments, one for the second and third, and so on. Straight-line segments can be generated by duplicating the end-points of the desired line segment. Set to false or {}, no smoothing is performed. In Tk 8.5+, set to raw, indicates that the line should also be drawn as a curve but where the list of coordinates is such that the first coordinate pair (and every third coordinate pair thereafter) is a knot point on a cubic Bezier curve, and the other coordinates are control points on the cubic Bezier curve.
-splinesteps *number*	Specifies degree of smoothness desired for curves by approximating spline over *number* line segments. Used with **-smooth** true or raw.

pathName **create oval** *?x1 y1 x2 y2? ?coordList? ?option value ...?*

Display an oval region. Args *x1*,*y1* and *x2*,*y2* or *coordList* give the coordinates of two diagonally opposite corners of a rectangular region enclosing the oval. The oval includes the top and left edges but not bottom and right edges. *Options* are:

-dash	**-activedash**	**-disableddash**	**-dashoffset**
-fill	**-activefill**	**-disabledfill**	
-outline	**-activeoutline**	**-disabledoutline**	
-outlinestipple	**-activeoutlinestipple**	**-disabledoutlinestipple**	
-stipple	**-activestipple**	**-disabledstipple**	**-offset**
-width	**-activewidth**	**-disabledwidth**	
-state			
-tags			

pathName **create polygon** *?x1 y1 ... xN yN? ?coordList? ?option value ...?*

Display a polygonal or curved filled region. Args *x1*,*y1* through *xN* ,*yN* or *coordList* give the coordinates of three or more points that define a polygon. The first point is not repeated as the last point. *Options* are:

-dash	**-activedash**	**-disableddash**	**-dashoffset**
-fill	**-activefill**	**-disabledfill**	
-outline	**-activeoutline**	**-disabledoutline**	
-outlinestipple	**-activeoutlinestipple**	**-disabledoutlinestipple**	
-stipple	**-activestipple**	**-disabledstipple**	**-offset**
-width	**-activewidth**	**-disabledwidth**	
-state			
-tags			

Polygon Specific Options:

-joinstyle *style*	(Tk 8.4+) Specifies joints to be drawn at outline verticies. *Style* options are: **bevel**, **miter**, or **round**.
-smooth *boolean*	Set to true, **bezier** smoothing is used to draw the outline as a curve. The line is rendered as a set of parabolic splines: one spline is drawn for the first and second line segments, one for the second and third, and so on. Straight-line segments can be generated by duplicating the end-points of the desired line segment. Set to false or {}, no smoothing is performed. In Tk 8.5+, set to raw, indicates that the line should also be drawn as a curve but where the list of coordinates is such that the first coordinate pair (and every third coordinate pair thereafter) is a knot point on a cubic Bezier curve, and the other coordinates are control points on the cubic Bezier curve.
-splinesteps *number*	Specifies degree of smoothness desired for outline curves by approximating spline over *number* line segments. Used with **-smooth** true or raw.

pathName **create rectangle** *?x1 y1 x2 y2? ?coordList? ?option value ...?*

Display a rectangular region. Args *x1*,*y1* and *x2*,*y2* or *coordList* give the coordinates of two diagonally opposite corners of the rectangle. The rectangle includes the top and left edges but not bottom and right edges. *Options* are:

-dash	**-activedash**	**-disableddash**	**-dashoffset**
-fill	**-activefill**	**-disabledfill**	
-outline	**-activeoutline**	**-disabledoutline**	
-outlinestipple	**-activeoutlinestipple**	**-disabledoutlinestipple**	
-stipple	**-activestipple**	**-disabledstipple**	**-offset**
-width	**-activewidth**	**-disabledwidth**	
-state			
-tags			

pathName **create text** *?x y? ?coordList? ?option value ...?*

Display a string of characters in one or more lines at positioning point coordinates of *x*,*y* or *coordList*. *Options* are:

-fill	**-activefill**	**-disabledfill**
-stipple	**-activestipple**	**-disabledstipple**
-state		
-tags		

Text Specific Options:

-anchor *anchorPos*	Specifies how to position the text relative to the item positioning point where *anchorPos* is **n**, **ne**, **e**, **se**, **s**,**sw** ,**w**,**nw**, or **center** (default).
-font *fontName*	Specifies the font to use for the text item. See Fonts for font options. Default is system dependent.
-justify *how*	Specifies how to justify multiple text lines within its bounding region. *How* options are: **left**, **right**, or **center**.
-text *string*	Specifies the characters to be displayed in the text item. Newline causes line break.
-width *lineLength*	Specifies the maximum line length for the text. If zero (default), break only on newline, otherwise break on last space before maximum line length. See Coordinates in Options and Resources for screen unit options.

pathName **create window** *?x y? ?coordList? ?option value ...?*	Display a window at positioning point coordinates of *x*,*y* or *coordList* . It is not possible to draw other graphical items on top of window items. A window item always obscures any graphics that overlap it, regardless of their order in the display list. *Options* are: **-state** **-tags** Window Specific Options: **-anchor** *anchorPos* — Specifies how to position the window relative to the item positioning point where *anchorPos* is **n**, **ne**, **e**, **se**,**s** ,**sw**,**w**,**nw**, or **center** (default). **-height** *height* — Height in screen units to assign item's window. See Coordinates in Options and Resources for screen unit options. **-width** *width* — Width in screen units to assign item's window. See Coordinates in Options and Resources for screen unit options. **-window** *pathName* — Specifies the window *pathName* to associate with this item. The window must be either a child of the canvas widget or a child of some ancestor of the canvas widget and not a top-level window.

3.4 Checkbutton

Command	Description
checkbutton *pathName ?options?*	Creates a checkbutton widget *pathName* with *options* and returns the new widget's path name. When invoked, *pathName* must not exist, but *pathName*'s parent should. A checkbutton widget displays a textual string, bitmap, or image and a square called an *indicator*. By default a checkbutton is configured to select and deselect itself on alternate button clicks. Each checkbutton monitors its associated variable and automatically selects and deselects itself when the variables value changes to and from the button's "on" value.Multiple fonts within a button text field are not supported.

Checkbutton Options

Standard

See Common Options and Resources in Options and Resources for full details.

-activebackground	**-disabledforeground**	**-padx**
-activeforeground	**-font**	**-pady**
-anchor	**-foreground**	**-relief**
-background	**-highlightbackground**	**-takefocus**
-bitmap	**-highlightcolor**	**-text**
-borderwidth	**-highlightthickness**	**-textvariable**
-compound (8.4+)	**-image**	**-underline**
-cursor	**-justify**	**-wraplength**

Checkbutton Specific

See Coordinates in Options and Resources for screen unit options.

Configure Option	Resource Name	Resource Class	Description
-command *script*	**command**	**Command**	Tcl command to associate with the button. *Script* is invoked when mouse button 1 is released over the button window. The button's global variable (**-variable** option) will be updated before the command is invoked.
-height *height*	**height**	**Height**	Height of button in screen units for bitmaps/images and in lines for text. Default is to auto size.
-indicatoron *boolean*	**indicatorOn**	**IndicatorOn**	Specifies whether the indicator should be drawn (default) or not. If **false**, the **relief** option is ignored and the relief is set to sunken when widget is selected and raised in all other cases.
-offrelief *type*	**offRelief**	**OffRelief**	(Tk 8.4+) Specifies the relief for the checkbutton when the indicator is not drawn and the checkbutton is off. Options are: **flat**, **raised** (default), and **sunken**.
-offvalue *value*	**offValue**	**Value**	Value (default is 0) stored in button's global variable when the checkbutton is deselected.
-onvalue *value*	**onValue**	**Value**	Value (default is 1) stored in button's global variable when the checkbutton is selected.
-overrelief *type*	**overRelief**	**OverRelief**	(Tk 8.4+) Alternative relief for when mouse cursor is over button. Not used when set to empty string (default). Options are: **flat**, **raised**, and **sunken**.
-selectcolor *color*	**selectColor**	**Background**	Specifies a background color to use when the button is selected. If set to empty string, no special color is used. If **-indicatoron** is true then the color applies to the indicator, if false this color is used as the background for the entire widget when selected.
-selectimage *image*	**selectImage**	**SelectImage**	Specifies image to be displayed when checkbutton is selected. Used with **-image**.
-state *state*	**state**	**State**	State of button. Options are: **active** (mouse pointer over button, use **activeforeground** and **activebackground**), **disabled** (button is insensitive, use **disabledforeground** and **background**), or **normal** (use **foreground** and **background**).
-tristateimage image	tristateImage	TristateImage	(Tk 8.5+) Specifies an image to display (in place of the image option) when the checkbutton is in tri-state mode. This option is ignored unless the image option has been specified.
-tristatevalue value	tristateValue	Value	(Tk 8.5+) Specifies the value that causes the checkbutton to display the multi-value selection, also known as the tri-state mode. Defaults to {}.
-variable *variable*	**variable**	**Variable**	Specifies name of global variable to use for button selection status. Default is name of the button within its parent.
-width *width*	**width**	**Width**	Width of button in screen units for bitmaps/images and in characters for text. Default is to auto size.

Effect	Options
Toolbar buttons	**-relief flat -overrelief raised**
Text-style toolbar buttons	**-offrelief flat -indicatoron false -overrelief raised**

Checkbutton Commands

Command	Description
pathName **cget** *option*	Returns the current value of the configuration *option* for checkbutton *pathName*. See Checkbutton Widget Options above for *options*.
pathName **configure***?option? ?value? ?option value ...?*	Change the configuration *option* for the checkbutton *pathNamevalue*. Without *value*, a list describing the available options is returned. Without *option*, a list describing all of the available options for *pathName* is returned. For multiple options an empty string is returned. See Checkbutton Widget Options to above for *options*.
pathName **deselect**	Deselect the checkbutton and set the associated variable to its "off" value.
pathName **flash**	Flash checkbutton by toggling between active and normal colors several times. Checkbutton is left is initial state of **active** or **normal**. Ignored if checkbutton is disabled.
pathName **invoke**	Toggle the selection state of the checkbutton and invoke the Tcl command specified with **-command**, if any. Returns value of Tcl command or empty string if no **-command**. Ignored if button is disabled.
pathName **select**	Select the checkbutton and set the associated variable to its "on" value.
pathName **toggle**	Toggle the selection state of the checkbutton, redisplaying it and modifying its associated variable to reflect the new state.

Default Checkbutton Bindings

Active or normal checkbutton default bindings:

Event	Description
<Enter>	On Unix, when mouse passes over button **state**becomes**active**.
<Leave>	On Unix, when mouse leaves the button **state** becomes **normal**.
<Button-1> or **<Return>** or **<space>**	On Unix, **relief** changes to **sunken** and associated **-command** *script* is executed.
<Button-1>	On Windows and Mac, **relief** changes to **sunken** and state becomes **active**.
<ButtonRelease-1>	On Windows and Mac, **relief** changes to **raised**, state becomes **normal**, and associated **-command** *script* is executed.
<Enter>	On Windows and Mac, **relief** changes to **sunken** and state becomes **active**.
<plus> or**<equal>**	On Windows, selects the button.
<minus>	On Windows, deselects the button.
<space>	On Windows and Mac, **relief** changes to **sunken** and associated **-command** *script* is executed.

3.5 Clipboard and Selection

The clipboard selection is a platform independent method that allows for the exchange of data between applications via copy, cut, and paste. Only X-Windows supports the use of other selection types for all applications. Tk understands all selection types on all platforms.

Command	Description
clipboard append*?***-displayof** *window*? ?**-format** *format? ?***-type** *type? ?--? data*	Append *data* to clipboard on *window* 's display in the form *type* with the representation *format* . Also claims ownership of clipboard on *window*'s display. See Common Target Atom Types below for *type* options. See Common Selection Property Types below for *format* options. *Format* is required for non-Tk clipboard users.
clipboard clear*?***-displayof** *window?*	Claim ownership of clipboard on *window*'s display (default is ".") and clears its contents.
clipboard get ?**-displayof** *window*? ?**-type** *type*?	(Tk 8.4+) Retrieve data from the clipboard on *window*'s display (default is ".") in form *type*. See Common Target Atom Types below for *type* options. Same as selection get -selection CLIPBOARD.
selection clear *?***-displayof** *window? ?***-selection** *selection?*	If *selection* (see Selection Atom Types below for options) exists anywhere on *window*'s display (default is "."), clear it so that no window owns the selection anymore.
selection get *?***-displayof** *window? ?***-selection** *selection? ?***-type** *type?*	Returns the value of *selection* (see Selection Atom Types below for options) from *window*'s display (default is ".") in format *type* (see Common Target Atom Types below for options). If the selection is returned in a non-string format, such as INTEGER or ATOM, the selection command converts it to string format as a collection of fields separated by spaces: atoms are converted to their textual names, and anything else is converted to hexadecimal integers.
selection handle*?***-selection** *selection? ?***-type** *type? ?***-format** *format? window command*	Creates a handler for selection requests, such that c*ommand* will be executed with args *offset* (starting char in selection) and *maxChars* (max chars to retreive) whenever *selection* (see Selection Atom Types below for options) is owned by *window* (default is ".") and someone attempts to retrieve it in the form given by *type* (see Common Target Atom Types below for options). *Format* (see Common Selection Property Types below for options) specifies how to format the data to the requestor. *Format* is required for non-Tk clipboard users. If *tclCommand* is an empty string, the existing handler is removed. Tk 8.4+ (broken in 8.4.0 and 8.4.1) adds a duplicate handler for UTF8_STRING when STRING is used for type.
selection own*?***-displayof** *window? ?***-selection** *selection?*	Returns the path name of the window in this application that owns *selection* (see Selection Atom Types below for options) on the display containing *window*, or an empty string if none.
selection own*?***-command** *command??***-selection** *selection? window*	Causes *window* to become the new owner of *selection* (see Selection Atom Types below for options) on *window*'s display and sets up a handler to run *command* when *window* loses the selection to another window later on.

Definitions

Clipboard	Platform independent method that allows for the exchange of data between applications via copy, cut, and paste.
Selection	Primary mechanism on X-Windows to exchange information via a copy and paste between clients. Selections are assigned to an particular atom such that other applications can retreive the selection by specifying the same atom.
Atom	Unique name (strings without a specific encoding) that clients can use to communicate information to each other.

Selection Atom Types

Selection Type	Description
PRIMARY	(default) Principal means of communication between clients on X-Windows that use the selection mechanism.
SECONDARY	Means of obtaining data when there is a primary selection and the user does not want to disturb it
CLIPBOARD	Used to hold data that is being transferred between clients usually for data that is being cut or copied and then pasted. This is the same buffer used by the clipboard command and is platform independent.
other	Client specific private atom.

Common Target Atom Types

Target Type	Description
ATOM	Converted into ATOM name.
FILE_NAME	The full path name of a file.
POSTSCRIPT	String data in postscript format.
STRING	(default) Text encoded in ISO Latin-1 character set plus tab and newline.
INTEGER	Converted to a collection of fields separated by spaces.
UTF8_STRING	Text encoded in UTF-8 character set plus tab and newline.
other	Converted to hexadecimal integers.

Common Selection Property Types

Selection Property Types	Description
ATOM	Fields are converted to 32-bit atom values separated by white-space.
STRING	(default) Uses 8-bit ASCII chars.

3.6 Console

Command	Description
console eval *script*	(Windows and Mac only) Evaluate the *script* argument as a Tcl script in the console interpreter.
console hide	(Windows and Mac only) Hide the console window from view.
console show	(Windows and Mac only) Display the console window. The console window replaces the real console for input and output on platforms that do not have a real console. It is implemented as a separate interpreter with the Tk toolkit loaded, and control over this interpreter is given through the **console** command.
console title ?*string* ?	(Windows and Mac only) Change name of console window to*title*. Without *string*, returns the console window title.
consoleinterp eval *script*	(Windows and Mac only) Evaluates *script* as a Tcl script at the global level in the main interpreter.
consoleinterp record *script*	(Windows and Mac only) Records and evaluates *script* as a Tcl script at the global level in the main interpreter as if *script* had been typed in at the console.

Default Console Bindings

In Tk 8.2.x+ all text bindings except **<Control-o>** and **<Control-v>** are also available. Tk 8.3.4 added numerous bindings from Tkcon.

Event	Description
<Tab>	Insert tab (/t) chanacter.
<Return>	Causes the current line to be passed to the main interpreter for evaluation.
<Delete>	Deletes the selected text (if any selected) or character right of the cursor.
<BackSpace>	Deletes the selected text (if any selected) or character left of the cursor.
<Control-a> or **<Home>**	Moves cursor to the start of the line after prompt.
<Control-e> or **<End>**	Moves cursor to the end of the line.
<Control-p> or **<Up>**	Selects the previous entry in the command history.
<Control-n> or **<Down>**	Selects the next entry in the command history.
<Control-b> or **<Left>**	Moves the cursor one character backwards (left) if not at prompt.
<Control-f> or **<Right>**	Moves the cursor one character forwards (right) if not at end of the line.
<Control-d>	Deletes the character to the right of the insertion cursor.
<Meta-d>	Deletes the word to the right of the insertion cursor.
<Control-k>	Deletes all the characters to the right of the insertion cursor.
<Control-t>	Reverses the order of the two characters to the right of the insertion cursor.
<Control-h> or **<Meta-BackSpace>**	Deletes the character to the left of the insertion cursor.
<F9>	Rebuilds console window by destroying all its children and reloading the Tcl script that defined the console's behaviour.
<Insert>	Inserts selected text into console window
<Keypress>	Insert character into entry widget.
<<Copy>>	Copy selected text to clipboard.
<<Cut>>	Works the same as **<<Copy>>** except selected text is deleted.
<<Paste>>	Paste text in clipboard to console window at cursor position.

3.7 Dialogs

Command	Description
tk_chooseColor *?option value ...?*	Creates a pop-up dialog box for the user to choose a color and returns the selected color. See Colors in Options and Resources for *color* formats. Options are:
-initialcolor *color*	Use *color* as the initial selected color.
-parent *window*	Makes *window* the parent of dialog.
-title *string*	Specifies the dialog window title.
tk_chooseDirectory *?option value ...?*	(Tk 8.3+) Creates a pop-up dialog box for the user to select a directory and returns the selected directory. Options are:
-initialdir *directory*	Use *directory* as initial directory. Default is current working directory. If initial directory is a relative path, the returned path will be the absolute path.
-mustexist *boolean*	Specifies whether only existing directories can be selected. Default is false.
-parent *window*	Makes *window* the parent of dialog.
-title *string*	Specifies the dialog window title.

tk_dialog *window title text bitmap default string ?string ...?*	Creates a pop-up modial dialog box, does a local grab, and waits for a response. *Window* is the top-level window to use (destroys window if it already exists). *Title* specifies the dialog window title. *Text* specifies the message to display in the dialog. *Bitmap* specifies the bitmap (See Default Bitmaps in Options and Resources) to display to the left of the message or no bitmap if set to an empty string. *Default* specifies the index of the default button (0 is the leftmost button) or no default if set to an empty string or negative number. Creates a button at the bottom of the dialog for each *string* arg. When done the dialog is destroyed and the index of the button selected is returned.
tk_getOpenFile *?option value ...?*	Creates a pop-up dialog box for the user to choose an existing filename and returns the choice. Non-existant files are rejected with an error prompt. Options are:
-defaultextension *extension*	String to append to filename if user enters a filename without an extension. Default is empty string or reasonable guess based on **-filetypes**, if specified.
-filetypes *filePatternList*	List of file types the user can choose from for determining which types of files to display, if supported by the platform. Format of elements: {{description {extensions ...} ?{MacTypes ...}?} ...}
-initialdir *directory*	Use *directory* as initial directory. Default is current working directory. If initial directory is a relative path, the returned path will be the absolute path.
-initialfile *fileName*	Specifies the default filename to be displayed in the dialog.
-multiple	(Tk 8.4+) Allows the user to choose multiple files from the Open dialog.
-message *string*	(Tk 8.4+) Specifies a message to include in the client area of the dialog on Macs.
-parent *window*	Makes *window* the parent of dialog.
-title *string*	Specifies the dialog window title.
tk_getSaveFile *?option value ...?*	Creates a pop-up dialog box for the user to choose a filename and returns the choice. If an existing file is selected, another pop-up is displayed to confirm the choice. Options are:
-defaultextension *extension*	String to append to filename if user enters a filename without an extension. Default is empty string or reasonable guess based on **-filetypes**, if specified.
-filetypes *filePatternList*	List of file types the user can choose from for determining which types of files to display, if supported by the platform. Format of elements: {{description {extensions ...} ?{MacTypes ...}?} ...}
-initialdir *directory*	Use *directory* as initial directory. Default is current working directory. If initial directory is a relative path, the returned path will be the absolute path.
-initialfile *fileName*	Specifies the default filename to be displayed in the dialog.
-message *string*	(Tk 8.4+) Specifies a message to include in the client area of the dialog on Macs.
-parent *window*	Makes *window* the parent of dialog.
-title *string*	Specifies the dialog window title.
tk_messageBox *?option value ...?*	Creates a message dialog with an application-defined message, an icon and a set of buttons. Returns the unique symbolic name of button pressed by the user. Not re-entrant, so multiple dialogs will interfere with each other. Options are:
-default *name*	Make button *name* the default. See **-type** for button names.
-detail string	Specifies an auxiliary message below -message in a less emphasized font (if available).
-icon *iconImage*	Specifies the icon to display. Options are: **error**, **info** (default), **question**, or **warning** .
-message *string*	Specifies the message to display in the message box.
-parent *window*	Makes *window* the parent of the message box.
-title *string*	Specifies the message box window title.
-type *buttonType*	Specifies which set of buttons to display. Options and symbolic names are: **abortretryignore** (**abort**,**retry**, and **ignore** buttons), **ok** (**ok** button), **okcancel** (**ok** and **cancel** buttons), **retrycancel** (**retry** and **cancel** buttons), **yesno** (**yes** or **no** buttons), or **yesnocancel** (**yes**, **no**, and **cancel** buttons). Default is **ok**.

3.8 Entry Widget

Command	Description
entry *pathName ?options?*	Creates an entry widget *pathName* with *options* and returns the new widget's path name. When invoked, *pathName* must not exist, but *pathName*'s parent should. An entry widget is used to display and/or allow alterations to one line of text.

Entry Options

Standard

See Common Options and Resources in Options and Resources for full details.

-background	**-highlightcolor**	**-relief**
-borderwidth	**-highlightthickness**	**-selectbackground**
-cursor	**-insertbackground**	**-selectborderwidth**
-disabledforeground (Tk 8.4+)	**-insertborderwidth**	**-selectforeground**
-exportselection	**-insertofftime**	**-takefocus**
-font	**-insertontime**	**-textvariable**
-foreground	**-insertwidth**	**-xscrollcommand**
-highlightbackground	**-justify**	

Entry Specific

Configure Option	Resource Name	Resource Class	Description
-disabledbackground *color*	**disabledBackground**	**DisabledBackground**	(Tk 8.4+) Background color of widget when the entry is disabled. If set to the empty string, the normal background color is used.
-invalidcommand *script*	**invalidCommand**	**InvalidCommand**	(Tk 8.3+) Specifies script to eval when **-validcommand** returns 0. If set to the empty string (default), disables option. Typically set to **bell**. See Percent Substitutions below for valid % substitutions. (Also **-invcmd**).
-readonlybackground *color*	**readonlyBackground**	**ReadonlyBackground**	(Tk 8.4+) Background color of widget when the entry is read-only. If set to the empty string, the normal background color is used.
-show *char*	**show**	**Show**	Show *char* instead of the actual characters for each character in entry.
-state *state*	**state**	**State**	State of entry. Options are: **disabled** (cannot change or select contents, uses **disabledforeground** and **background**), **normal** (can change and select contents, uses **foreground** and **background**), or **readonly** (Tk 8.4+, cannot change but can select contents).
-validate *mode*	**validate**	**Validate**	(Tk 8.3+) Specifies validation mode. See Validation Types below for options.
-validatecommand *script*	**validateCommand**	**ValidateCommand**	(Tk 8.3+) Specifies script to eval when entry input is to be validated. If set to the empty string (default), disables option. Script must return 1 to accept or 0 to reject new value. See Percent Substitutions below for valid % substitutions. (Also **-vcmd**).
-width *width*	**width**	**Width**	Width of entry window in font average-sized characters. If <=0, auto size based on current text.

Validation Types

Type	Description
none	Do not perform validation (default).
focus	**-validatecommand** will be called when the entry receives or loses focus.
focusin	**-validatecommand** will be called when the entry receives focus.
focusout	**-validatecommand** will be called when the entry loses focus.
key	**-validatecommand** will be called when the entry is edited.
all	**-validatecommand** will be called for all above conditions.

Percent Substitutions

Form	Description
%d	(Tk 8.3+) Type of action: 1 for **insert**, 0 for **delete**, or -1 for focus, forced, or textvariable validation.
%i	(Tk 8.3+) Index of char string to be inserted/deleted, if not -1.
%P	(Tk 8.3+) The value of the entry should **-validatecommand** accept the new entry. When configuring to a new textvariable, this will be the value of that textvariable.
%s	(Tk 8.3+) The current value of entry before **-validatecommand** accepts the new entry.
%S	(Tk 8.3+) The text string being inserted/deleted, if not an empty string {}.
%v	(Tk 8.3+) The current validation type (**none**, **focus**, **focusin**,**focusout**, **key**, or **all**).
%V	(Tk 8.3.1+) The type of validation that triggered the callback (**key**, **focusin**, **focusout**, **forced**).
%W	(Tk 8.3+) The name of the entry widget.

Indicies or Character Positions

Some entry commands support the use of an index to locate the position of characters within the entry string starting from 0. The following are the valid forms of specifying an *index*:

Index form	Description
number	A decimal number giving the position or index (starting from 0) of the desired character within the entry string. If *number* < 0, the 0 is used, if *number* > length of text list, then **end** is used.
anchor	Selection anchor point as set by the **select from** and **select adjust** commands.
end	Character or coordinate just after last one in entry's string.
insert	Character just after the insertion cursor.
sel.first	First character in selection.
sel.last	Character just after last character in selection.
@number	Character at the x-coordinate point in the entry's window. If *x* is outside the entry window's range, it is set to the nearest legal value.

Entry Widget Commands

Command	Description
pathName **bbox***index*	Returns a list of four elements *x y w h* , giving an approximate bounding box for the character at position *index* . Coordinates *x,y* are top-left corner of character at *index*,*w* is width of char, and *h* is height of char in pixels.
pathName **cget** *option*	Returns the current value of the configuration *option*. See Entry Widget Options above for *options* .
pathName **configure** *?option? ?value? ?option value ...?*	Change the configuration *option* to *value*. Without *value*, a list describing *option* is returned. Without *option*, a list of all available options for *pathName* is returned. For multiple options an empty string is returned. See Entry Widget Options above for *options*.
pathName **delete** *first ?last?*	Delete characters in entry's string from position *first* up to but not including position *last* (default is *first*+1 to delete 1 character). See Indicies or Character Positions above for *first* and *last* options.
pathName **get**	Returns the entry's string.
pathName **icursor** *index*	Display the insertion cursor just before the character at position *index*. See Indicies or Char Positions above for index options.
pathName **index** *index*	Returns the numerical index corresponding to *index*. See Indicies or Character Positions above for index options.
pathName **insert** *index string*	Insert *string* just before the character at position *index*. See Indicies or Character Positions above for index options.
pathName **scan** *option args*	Implements scanning on entry widgets. *Options* are:
mark *x*	Records *x* and the current view in the entry window. Typically associated with mouse button press in widget.
dragto *x*	Adjusts the view by 10 times the difference between the coordinate *x* and the last **mark** *x* coordinate. Used with mouse motion events to produce high speed dragging.
pathName **selection** *option arg*	Manipulates the selection within an entry based on *option*. See Indicies or Char Positions above for *index* options. Vaild *options* and *args* are:
adjust *index*	Adjust the end of the selection nearest to the character given by position *index* to include characters up to *index* and set the other end to be the anchor point. Works the same as **selection to** if selection is not in entry widget.
clear	Clear the selection if it is in the widget.
from *index*	Sets the selection anchor point to the character just before position *index*.
present	Returns 1 if characters are selected in the entry, 0 if not.
range *start end*	Sets the selection to include characters from position *start* up to but not including position *end*.
to *index*	If *index* < anchor point, set the selection to include characters from position *index* up to but not including the anchor point. If *index* > anchor point, set the selection to include characters from the anchor point up to but not including position *index*. If *index* = anchor point, no change is made. If the selection isn't in the entry widget, use the most recent anchor point specified for the widget.
*pathName***validate**	(Tk 8.3+) Forces the evaluation of **-validatecommand** by temporarily setting **validate** to **all** and returns result.
pathName **xview** ?*option args*?	Query or change the horizontal entry widget view. Without any *options*, returns a two element list specifying the start and end of the visible fraction (from 0 to 1) of the horizontal span of the widget between the left and right edges of the window. Vaild *options* and *args* are:
index	Adjust window view to display the character at position *index* at the left edge of window. See Indicies or Char Positions above for *index* options.
moveto *fraction*	Adjust window view so that *fraction* (from 0 to 1) of the total width of the widget is off-screen to the left.
scroll *number* **pages**	Shift the view left (*number* < 0) or right (*number* > 0) by *number* screenfuls.
scroll *number* **units**	Shift the view left (*number* < 0) or right (*number* > 0) by *number* average-width characters.

Default Entry Widget Bindings

For additional default bindings see Virtual Events in Bindings and Virtual Events.

Event	Description
<Button-1>	Positions the insertion cursor just before the character underneath the mouse cursor, sets the input focus to this widget, and clears any selection in the widget.
<B1-Motion>	Drags out a selection (in words if double clicked) between the insertion cursor and the character under the mouse.
<Double-Button-1>	Selects the word under the mouse and positions the insertion cursor at the beginning of the word.
<Triple-Button-1>	Selects all of the text in the entry and positions the insertion cursor before the first character.
<Shift-B1-Motion>	Adjusts the end of the selection (in words if double clicked) that was nearest to the mouse cursor when button 1 was pressed.
<Control-Button-1>	Position the insertion cursor in the entry without affecting the selection.
<B1-Leave>	Adjusts view in entry left or right more quickly.
<B1-Enter>	Stops adjustment of view in entry left or right more quickly.
<Button-2>	Paste selection into the entry at the position of the mouse cursor.
<B2-Motion>	Adjusts view in entry by scrolling left or right.
<Left> or **<Control-b>**	Moves the insertion cursor one character back (left), clears any selection in the entry, and sets the selection anchor.
<Right> or **<Control-f>**	Moves the insertion cursor one character forward (right), clears any selection in the entry, and sets the selection anchor.
<Shift-Left>	Move the insertion cursor one character back (left) and extend the selection to include the new character.
<Shift-Right>	Move the insertion cursor one character forward (right) and extend the selection to include the new character.
<Control-Left> or **<Meta-b>**	Move the insertion cursor back (left) by one word, clears any selection in the entry, and sets the selection anchor.
<Control-Right> or **<Meta-f>**	Move the insertion cursor forward (right) by one word, clears any selection in the entry, and sets the selection anchor.
<Shift-Control-Left>	Move the insertion cursor back (left) by one word and also extend the selection.
<Shift-Control-Right>	Move the insertion cursor forward (right) by one word and also extend the selection.
<Home> or **<Control-a>**	Move the insertion cursor to the beginning of the entry and clear any selection in the entry.
<Shift-Home>	Move the insertion cursor to the beginning of the entry and also extends the selection to that point.
<End> or **<Control-e>**	Move the insertion cursor to the end of the entry and clear any selection in the entry.
<Shift-End>	Move the insertion cursor to the end of the entry and also extends the selection to that point.
<Select> or **<Control-Space>**	Set the selection anchor to the position of the insertion cursor without affecting the selection.
<Shift-Select> or **<Shift-Control-Space>**	Adjusts the selection to the current position of the insertion cursor, if there is one, otherwise it selects from the anchor to the insertion cursor.
<Control-slash>	Selects all the text in the entry.
<Control-backslash>	Clears any selection in the entry.
<Delete>	Deletes the selection, if there is one, otherwise it deletes the character to the right of the insertion cursor.
<BackSpace> or **<Control-h>**	Deletes the selection, if there is one, otherwise it deletes the character to the left of the insertion cursor.
<Control-d>	Deletes the character to the right of the insertion cursor.
<Meta-d>	Deletes the word to the right of the insertion cursor.

<Control-k>	Deletes all the characters to the right of the insertion cursor.
<Control-t>	Reverses (transposes) the order of the two characters to the right of the insertion cursor.
<Keypress>	Insert character into entry widget.
<<Copy>>	Copy selected text to clipboard.
<<Cut>>	Works the same as **<<Copy>>** except selected text is deleted.
<<Paste>>	Paste text in clipboard to console window at cursor position.

3.9 Fonts

Command	Description
font actual *font* ?**-displayof** *window*? ?*option*?	Returns actual value for *font*'s *option* on *window*'s (default is main window) display. Without *option*, a list of all option and value pairs is returned. See Font Descriptions and Font Options below for *font* and *option* values, respectively.
font configure *fontname ?option? ?value? ?option value ...?*	Sets each *option* to specified *value* for *fontname*. Without *value*, the current value of *option* is returned. Without *option*, a list of all option and value pairs is returned. For multiple options an empty string is returned. See Font Options below for *option*s.
font create *?fontname? ?option value ...?*	Create a new font *fontname* and returns the font name. Without *fontname* default naming convention is **font**# where # is an integer. See Font Options below for *options*.
font delete *fontname ?fontname ...?*	Delete all of the specified fonts. Does not remove font if it is in use by a widget until all instances are released.
font families ?**-displayof** *window* ?	Returns a list of all font families defined on *window*'s display (default is main window).
font measure *font* ?**-displayof** *window? text*	Returns width of string *text* (except /n and /t) in pixels using *font* in *window* (default is main window). See Font Descriptions below for *font*.
font metrics *font* ?**-displayof** *window? ?option?*	Returns value of *font*'s metric *option* on *window*'s (default is main window) display. Without *option*, a list of all option and value pairs is returned. See Font Metrics and Font Options below for valid font metrics and*option* values, respectively.
font names	Returns list of currently defined fonts with names.

Font Description

The valid forms for the *font* options above are as follows. The form used is the first match meeting the match criteria.

#	Font name	Match	Description
1.	*fontname*	Exact only	Name of font created using **font create**. When used, won't cause error even if corresponding attrributes are invalid. If font with exact attributes can't be displayed, another close font will be substituted automatically.
2.	*systemfont*	Exact only	Name of platform-specific font interpreted by graphics server. See Platform Specific Fonts below.
3.	*family ?size? ?style ...?*	Closest match	A list where the first element the font *family* name, the optional second element is desired size (See **-size** in Font Options), and the optional *style* options are: **normal** or **bold**, **roman** or **italic** ,**underline,** and **overstrike**.
4.	X-font name	Closest match	A Unix-centric font name of the form of: *-foundry-family-weight-slant-setwidth-addstyle-pixel-point-resx-resy-spacing-width-charset-encoding* . The "*****" character may be used to skip individual fields and an individual "*" must be used for each skipped field except at the end.
5.	*option value ?option value ...?*	Closest match	A list of *option* and *value* pairs specifying the font options in the same format as **font create** . See Font Options below for *option*s.

Font Options

Option	Description
-family *name*	Specifies case-insensitive font family *name* . See Default Cross-Platform Fonts below for supported *name*s.
-size *size*	Specifies font *size* in points (or pixels if negative). If invalid, a close size will be used. A size of 0 uses the platform specific default.
-weight *weight*	Specifies font thickness as either **normal** (default) or **bold**.
-slant *slant*	Specifies whether the font is **roman** (default) or **italic**.
-underline *boolean*	Specifies whether font is underlined or not (default).
-overstrike *boolean*	Specifies whether font is overstriked or not (default).

Font Metrics

The valid **font metric** options are as follows. The baseline of a font is the horizontal line where the bottom of most letters (without descenders) line up.

Metric	Description
-ascent	Returns the distance in pixels that the tallest letter sticks up above the baseline of the font, plus any extra blank space added by the designer of the font.
-descent	Returns the distance in pixels that any letter sticks down below the baseline of the font, plus any extra blank space added by the designer of the font.
-linespace	The vertical distance in pixels between the baseline of two lines of text using the same font so that characters do not overlap. Usually this is the sum of the ascent above the baseline line plus the descent below the baseline.
-fixed	Returns a 1 is the font is fixed-width or 0 if it is proportionally-spaced.

Default Cross-Platform Fonts

Default font family names with fonts guaranteed to be supported by Tk denoted by (*).

Avant Garde	**Courier New**	**New Century Schoolbook**	**Times** (*)
Arial	**Geneva**	**New York**	**Times New Roman**
Bookman	**Helvetica** (*)	**Palatino**	**Zapf Chancery**
Courier (*)	**Monaco**	**Symbol**	**Zapf Dingbats**

System Specific Fonts

X Windows:

All valid X font names, including those listed by xlsfonts, are available.

MS Windows:

ansi	**ansifixed**	**device**	**oemfixed**	**system**	**systemfixed**

Mac:

system	**application**

3.10 Frame Widget

Command	Description
frame *pathName ?options?*	Creates a frame *pathName* with *options* and returns the new widget's path name. A frame widget is used as a spacer or container for complex window layouts.

Frame Options

Standard

See Common Options and Resources in Options and Resources for full details.

-borderwidth	**-highlightcolor**	**-pady** (Tk 8.4+)
-cursor	**-highlightthickness**	**-relief**
-highlightbackground	**-padx** (Tk 8.4+)	**-takefocus**

Frame Specific

See Coordinates in Options and Resources for screen unit options.

Configure Option	Resource Name	Resource Class	Description
-background *color*	**background**	**Background**	Same as standard **-background** expect if set to empty string, the widget will not display or allocate a colormap entry for the background or border color.
-class *name*	**class**	**Class**	Specifies class name to use in querying the option database and for bindings. Can not be changed with **configure** command.
-colormap *colormap*	**colormap**	**Colormap**	Specifies colormap (default is same as parent) to use for the window where *colormap* can be **new** (allocate new colormap) or the name of another window on same display with same visual. Can not be changed with **configure** command.
-container *boolean*	**container**	**Container**	Specifies whether the frame will be a container to embed another application. Can not be changed with **configure** command.
-height *height*	**height**	**Height**	Height of frame in screen units.
-visual *visual*	**visual**	**Visual**	Specifies the visual to use for the window. Default is the same as the parent. See Screen or Window Visuals in Toplevel for *visual* options. Can not be changed with **configure** command.
-width *width*	**width**	**Width**	Width of frame in screen units.

Frame Commands

Command	Description
pathName **cget** *option*	Returns the current value of the configuration *option*. See Frame Options above for *options*.
pathName **configure** *?option? ?value? ?option value ...?*	Change the configuration *option* to*value*. Without *value*, a list describing *option* is returned. Without *option*, a list of all available options for *pathName* is returned. For multiple options an empty string is returned. See Frame Options above for *options*.

3.11 Geometry Mangement

Grid

Geometry manager that arranges widgets in a grid (rows and columns) inside of another window, called the geometry master (or master window). See Coordinates in Options and Resources for screen unit options. Don't pack and grid children into the same widget (same level) unless geometry propagation is turned off.

Command	Description
grid *slave ?slave ...? ?option value ...?*	Same as **grid configure** *slave*.
grid anchor master ?anchor?	(Tk 8.5+) Specifies how to place the grid within the master when no row/column has any weight. Valid *anchor* values: **n**, **ne**, **e**, **se** ,**s** ,**sw** ,**w** ,**nw**, or **center** (default).
grid bbox *master ?column row? ?column2 row2?*	Returns a 4 element list describing the bounding box in pixels of the space occupied by area spanning between given cells. With only *column* and *row* , returns bounding box for specified cell (top left cell is 0). Without options, returns bounding box of grid as a list of integers (column1 row1 column2 row2).
grid columnconfigure *master index ?option value ...?*	Set or query column properties of *index* column(s) in grid *master*. *Index* may be a list of column indicies. Without *value*, the current value is returned for *option*. Witout *option*, all current *option* and *value* pair settings are returned. In Tk 8.5+, index can be all in order to apply to all columns.

-minsize *size*	Minimum size of *index* column(s) in screen units.
-pad *amount*	Padding in screen units to add to the largest window contained completly in column(s) *index* when grid requests their sizes.
-uniform *value*	(Tk 8.4+) Places the column in a uniform group with other columns that have the same *value* for **-uniform**. Not used if set to empty value.
-weight *int*	Relative weight for apportioning extra space among columns. Columns with a weight of 0, will not deviate from requested size.
grid configure*slave ?slave ...? ?option value ...?*	Set or query how *slave* windows should be managed by the grid geometry master. Unless the *slave* was previously managed, options not specified will be set to their default values. See Grid Relative Placement below for alternate *slave* options.
-column *n*	Insert the slave so that it occupies the *n* th column in the grid. Default is just to right of previously specified slave or 0 if none. Column numbers start with 0. Each use of "**x**" preceding *slave* increments the column position by 1.
-columnspan *n*	Insert the slave so that it occupies *n* columns (default is 1) in the grid. Each use of "**–**" following the *slave* name increments the column span by one.
-in *other*	Insert the *slave*(s) in the master window given by *other* (default is first *slave*'s parent window).
-ipadx *amount*	Specifies *amount* (default is 0), in screen units, of horizontal internal (added to border) padding to leave on each side of *slave*(s).
-ipady *amount*	Specifies *amount* (default is 0), in screen units, of vertical internal (added to border) padding to leave at the top and bottom of *slave*(s).
-padx *amount*	Specifies horizontal external (outside of border) padding *amount* (default is 0), in screen units to leave on each side of *slave*(s). In Tk 8.4+, *amount* may be a two element list consisting of left and right padding values.
-pady *amount*	Specifies vertical external (outside of border) padding *amount* (default is 0), in screen units to leave at the top and bottom of *slave*(s). In Tk 8.4+, *amount* may be a two element list consisting of top and bottom padding values.
-row *n*	Insert the slave so that it occupies the *n*th row in the grid. Default is same as previously specified slave or first unoccupied row if none. Row numbers start with 0.
-rowspan *n*	Insert the slave so that it occupies *n* rows (default is 1) in the grid. If the next **grid** command contains "^" characters for the same row as *slave*(s), then the **rowspan** of *slave* is extended by one. The number of ^'s in a row must match the number of columns spanned by the slave above it.
-sticky *style*	Specifies where to position a slave within the cell if the cell is larger than the requested dimensions. *Style* can be zero or more positions (**n**, **s**,**e** or **w)** with optional space and comma separators. If both **n** and **s** (or **e** and **w**) are specified, the slave will be stretched to fill the entire height (or width) of its cavity. The default or when set to an empty string, is to center the slave within the cell.
grid forget*slave ?slave ...?*	Removes and unmaps each *slave* from grid and forgets their configuration options.
grid info*slave*	Returns a list of option and value pairs describing the configuration state of *slave*. The first two elements are "**-in** *master*" where *master* is the slave's master.
grid location*master x y*	Returns column and row containing screen units *x* and *y* in *master*. Returns -1 if *x* or *y* is above or to the left of the grid.
grid propagate*master ?boolean?*	Specifies whether *master* tries to resize its slave windows to fit grid (default) or not. Without *boolean*, returns current setting.
grid remove*slave ?slave ...?*	Removes and unmaps each *slave* from grid and remembers their configuration options.
grid rowconfigure*master index ?option value ...?*	Set or query row properties of *index* row(s) in grid *master*. *Index* may be a list of row indicies. Without *value*, the current value is returned for *option*. Witout *option*, all current *option* and *value* pair settings are returned. See Grid Relative Placement below for alternate *slave* options. In Tk 8.5+, index can be all in order to apply to all rows.

-minsize *size*	Minimum size of *index* row(s) in screen units.
-pad *amount*	Padding in screen units to add to the largest window contained completly in row(s) *index*when grid requests their sizes.
-uniform *value*	(Tk 8.4+) Places the row in a *uniform group* with other rows that have the same *value* for **-uniform**. Not used if set to empty value.
-weight *int*	Relative weight for apportioning extra space among rows. Rows with a weight of 0, will not deviate from requested size.
grid size*master*	Returns size of grid in columns and rows for *master*.
grid slaves*master ?options?*	Returns a list of the slaves in *master* for the specified *column* and/or *row* . Without options, all slaves are returned.
-column *column*	Only return slaves in colum column.
-row *row*	Only return slaves in row row.

Grid Relative Placement

The **grid** command supports a limited capability to create layouts without specifying the row and column information for each slave. In this case, **grid** chooses default values for **column**, **row**, **columnspan**, and **rowspan** at the time the *slave* is managed based on the current grid layout, the position of the *slave* relative to other *slave*s in the same **grid** command, and the presence of the symbols **-**,**x**, and ^ with the *slave* names. When the symbol is repeated, the effect is also repeated.

Symbol	Effect
-	Increases **columnspan** of slave to the left.
x	Leave an empty column.
^	Extends the **rowspan** of slave above.

Pack

Geometry manager that arranges the children (slaves) of a parent (master) by packing them in order (defined by packing list) in the packing cavity around the edges of the parent. The packer allocates a rectangular *parcel* for the slave along the side of the cavity given by the slave's **-side** option. See Coordinates in Options and Resources for screen unit options.

Command	Description
pack *slave ?slave ...? ?options?*	Same as **pack configure**.
pack configure *slave ?slave ...? ?option value ...?*	Sets how *slave* windows should be managed by the packer. Valid *options* are:
-after *sibling*	Insert *slaves* after widget *sibling* in the packing order of *sibling*'s master.
-anchor *anchor*	Where to position *slave* in master window when smaller then the allocated space. Valid *anchor* values: **n**, **ne**, **e**, **se** ,**s** ,**sw** ,**w** ,**nw**, or **center** (default).
-before *sibling*	Insert *slaves* before widget *sibling* in the packing order of *sibling*'s master.
-expand *boolean*	Specifies whether the slaves should expand to consume extra space in their master or not (default).
-fill *style*	Specifies whether slaves should be stretched if the allocated space is larger than the requested dimensions. Options are: **none** (use requested dimensions plus internal padding), **x** (stretch slave horizontally to fill allocated space with room left over for padding), **y** (stretch slave vertically to fill allocated space with room left over for padding), or **both** (do both **x** and **y** stretching).
-in *master*	Insert *slave* at the end of the packing order in window *master*.
-ipadx *amount*	Specifies *amount* of horizontal internal padding to leave on each side of *slave* in screen units (default is 0).
-ipady *amount*	Specifies *amount* of vertical internal padding to leave on each side of *slave* in screen units (default is 0).
-padx *amount*	Specifies *amount* of horizontal external padding to leave on each side of *slave* in screen units (default is 0). In Tk 8.4+, *amount* can be a list of two values for the left and right side padding.
-pady *amount*	Specifies *amount* of vertical external padding to leave on each side of *slave* in screen units (default is 0). In Tk 8.4+, *amount* can be a list of two values for the top and bottom side padding.
-side *side*	Specifies which side of the master the slave(s) will be packed against. Options are **left**, **right**,**top**, or **bottom**.
pack forget *slave ?slave ...?*	Removes and unmaps each *slave* from the packing order and forgets their configuration options.
pack info *slave*	Returns a list of option and value pairs describing the configuration state of *slave* . The first two elements are "**-in** *master*" where *master* is the slave's master.
pack propagate *master ?boolean?*	Specifies whether window *master* tries to resize its slave windows for geometry propagation (default) or not. Without *boolean*, returns current setting.
pack slaves *master*	Returns a list of the slaves in the packing order for window *master*. If none, empty string is returned.

Place

Geometry manager for fixed placement, where the size and location of slave windows is user specified within another window called the master. The placer also provides rubber-sheet placement, where the user specifies the size and location of the slave in terms of the dimensions of the master, so that the slave changes size and location in response to changes in the size of the master. The placer supports mixing both styles of placement for slaves.

Command	Description
place*window option value ?option value ...?*	Same as **place configure**.
place configure *window ?option? ?value? ?option value ...?*	Change the configuration *option* to*value* for the slave given by *window*. Without *value*, a list describing *option* is returned. Without *option*, a list of all available options for *pathName* is returned. For multiple options an empty string is returned. Valid options are:
-anchor *anchor*	*Anchor* specifies which point of *window* is to be positioned at coordinate x,y as defined by **-relx**, **-rely** ,**-x**,**and -y**. Valid *anchor* values: **n**, **ne** , **e**,**se**,**s**,**sw**,**w**,**nw** (default), or**center**.
-bordermode *style*	Specifies the degree to which borders within the master are used in determining the placement of the slave. Options are: **inside** (default) where placer only uses innermost area of master inside any border, **outside** where placer considers area to include border, or **ignore** where placer ignores borders and like X windows includes internal border but not external border.
-height *size*	Specifies the height, including border, for*window* in screen units. Default or when set to empty string is to auto size.
-in *master*	Specifes the path name of the window relative to which *window* is to be placed. *Master* must be the parent or descendent *window*'s parent and in the same top level window. Default is to use *window*'s parent.
-relheight *size*	Specifies the floating point (0 to 1) height relative to the master. If used with **-height**, both values are summed before use.
-relwidth *size*	Specifies the floating point (0 to 1) width relative to the master. If used with **-width**, both values are summed before use.
-relx *location*	Specifies the floating point x-coordinate of the anchor point for *window* relative to the master, where 0 is the left edge and 1 is the right edge. *Location* need not lie within the bounds of the master window. If used with **-x**, both values are summed before use.
-rely *location*	Specifies the floating point y-coordinate of the anchor point for *window* relative to the master, where 0 is the top edge and 1 is the bottom edge. *Location* need not lie within the bounds of the master window. If used with **-y**, both values are summed before use.
-width *size*	Specifies the width, including border, for *window* in screen units. Default or when set to empty string is to auto size.
-x *location*	Specifies the x-coordinate in screen units of the anchor point for *window* in master.*Location* need not lie within the bounds of the master window.
-y *location*	Specifies the y-coordinate in screen units of the anchor point for *window* in master. *Location* need not lie within the bounds of the master window.
place forget*window*	Placer will stop managing and unmap *window*.
place info*window*	Returns a list of option and value pairs describing the configuration state of *window*. In Tcl 8.4.2+, the first two elements are "**-in** *master*" where *master* is the window's master.
place slaves*master*	Returns a list of the slaves for window *master*. If none, empty string is returned.

3.12 Images

Image Commands

Command	Description
image create *type* ?*name*? ?*options value* ...?	Creates new image *name* (default is **image#** where # is an integer) of *type* with *options* and returns the path name. If *name* already exists, it is replaced. *Type* can be either **bitmap** or **photo**. See Bitmap Image Options or Photo Image Options below for *options*. Don't use the same name as an existing command, or the command will be overwritten.
image delete *?name ...?*	Deletes each of the image *name*s. If an image is in use by a widget, it won't be deleted until all instances are released. Deleteing a widget using an image does not delete the image.
image height *name*	Returns height of image *name* in pixels.
image inuse *name*	(Tk 8.4+) Returns 0 if image *name* is in use by a widget, or 1 if not.
image names	Returns a list of of all the existing image names.
image type *name*	Returns the type (**bitmap** or **photo**) of image *name*.
image types	Returns a list of valid image types (Tk defaults are: **bitmap** and **photo**).
image width *name*	Returns width of image *name* in pixels.

The Bitmap Image Type

A bitmap is an image whose pixels can be either one of two colors or transparent. A bitmap image consists of identically sized background color, foreground color, source, and mask bitmaps. Each bitmap consists of 0/1 values in a rectangular array of pixels. If the mask for a pixel is 0, the image displays nothing (transparent effect) otherwise the source bitmap pixel is used. If the source for a pixel is 1, the foreground color is shown, otherwise the background color is shown. The options for **image create bitmap** are:

Bitmap Image Options	Description
-background *color*	Set background *color* for bitmap. If set to an empty string, the background pixels will be transparent.
-data *string*	Specify contents of bitmap in X11 **bitmap** program format as a string. Takes precedence over **-file**.
-file *fileName*	Use *fileName* as the source of the bitmap image. The bitmap must be in X11 **bitmap** program format.
-foreground *color*	Set foreground *color* for bitmap.
-maskdata *string*	Specify contents of mask in X11 **bitmap** program format as a string. Takes precedence over **-maskfile**.
-maskfile *fileName*	Use *fileName* as the source of the mask image. The bitmap must be in X11 **bitmap** program format.

Bitmap Image Command	Description
imageName **cget***option*	Returns the current value of the configuration *option* for image *imageName*. See Bitmap Image Options above for *options*.
imageName **configure***?option? ?value? ?option value ...?*	Change the configuration *option* for the image *imageName* to *value*. Without *value*, a list describing the option is returned. Without *option*, a list describing all of the available options for *imageName* is returned. For multiple options an empty list is returned. *Option* may have any of the *values* accepted by the **image create bitmap** command. See Bitmap Image Options above for *options*.

The Photo Image Type

A photo is an image whose pixels can display any color or be transparent. A photo image is stored internally in full color (32 bits per pixel), and is displayed using dithering if necessary. At present, standard Tk only supports the GIF, PPM, and PGM formats without an extension. The IMG extension adds support for: BMP, XBM, XPM, GIF (with transparency, but without LZW), PNG, JPEG, TIFF, and postscript. A photo image is transparent in regions where no image data has been supplied or where it has been set transparent by the transparency set subcommand. The options for **image create photo** are:

Photo Image Options	Description
-data *string*	Specify contents of image as a string in a supported format. The string can contain base64 encoded data or binary data. Takes precedence over **-file**. Supports PGM and PPM in Tk 8.4.7+.
-format *formatName*	Specify format for data specified with the **-data** or **-file** options. The **gif**, pgm, and ppm formats are supported for reads and **gif87**, **gif89**, pgm, and ppm formats are supported for writes.
-file *fileName*	Use *fileName* as the source of the photo image for a supported format.
-gamma *value*	(Tk 8.4+) Correct the colors allocated for displaying this image for a non-linear display with the specified gamma exponent *value*. *Value* must be > 0, default is 1 (no correction). *Value* > 1 will make image lighter, *Value* < 1 will make image darker.
-height *height*	Specifies the height of the image, in *height*pixels. Use 0 (default) to allow the image to expand or shrink vertically to fit the data.
-palette *paletteSpec*	Set the resolution of the color cube (number of colors) to be allocated for image. String *paletteSpec* can be a single decimal number to specify the number of shades of gray to use (monochrome), or three decimal numbers separated by slashes (/), to specify the number of shades of red, green and blue to use, respectively.
-width *width*	Specifies the width of the image, in *width*pixels. Use 0 (default) to allow the image to expand or shrink horizontally to fit the data.

The commands that write data to the image can expand the size of the image if necessary unless **-width** and/or **-height** are specified to prevent changing the image size. The following are the valid commands for photo images:

Photo Image Command	Description
imageName **blank**	Blanks the image so has no data and is completely transparent.
imageName **cget***option*	Returns the current value of the configuration *option* for image *imageName*. See Photo Image Options above for *options*.
imageName **configure** *?option? ?value? ?option value ...?*	Change the configuration *option* for the image *imageName* to *value*. Without *value*, a list describing the option is returned. Without *option*, a list describing all of the available options for *imageName* is returned. For multiple options an empty string is returned. See Photo Image Options above for *options*.
imageName **copy** *sourceImage ?option value ...?*	Copy a region from *sourceImage* to *imageName* using given options.

-compositingrule *rule*	(Tk 8.4+) Specifies how transparent pixels in *sourceImage* are combined with *imageName*. Rule **overlay** (default) specifies the *sourceImage* should be overlayed on *imageName*. Rule **set** specifies *imageName* be replaced by *sourceImage*.
-from *x1 y1 x2 y2*	Specifies rectangular sub-region (default is whole image) of the image in *sourceImage* to copy into *imageName* where (*x1*,*y1*) is the top left and (*x2*,*y2*) is the bottom right (or bottom right corner if not specified). Includes the left and top edges but not the bottom or right edges.
-shrink	Will shrink image in *sourceImage* so it fits within the current bottom-right corner of *imageName* without affecting the **image create** settings for **-height** or **-width**.
-subsample *x y*	Reduces source region of *sourceImage* by using only every *x*th and *y*th pixel in respective direction when copying to *imageName*. Negative values will cause image to be flipped about the respective axis. If not specified, *y* defaults to same value as *x*.
-to *x1 y1 x2 y2*	Specifies rectangular sub-region of *imageName* into which *sourceImage* will be copied with tiling if necessary, where (*x1*,*y1*) is the top left and (*x2*,*y2*) is the bottom right (or bottom right corner if not specified). Without *x2* and *y2*, the default is *(x1*,*y1*) plus size of *sourceImage*.
-zoom *x y*	Magnifies source region in *sourceImage* by *x* and *y* in respective direction in *imageName*. If not specified, *y* defaults to same value as *x*. Both *x* and *y* must be > 0.
imageName **data** *?option value ...?*	(Tk 8.3+) Returns image data in the form of a string. Options are:
-background *color*	If specified, all transparent pixels will be replaced with *color*.
-format *formatName*	Specify format for *imageName* (default is auto select). The **GIF**, **PGM**, **PPM** formats are supported.
-from *x1 y1 x2 y2*	Specifies rectangular sub-region (default is whole image) of the image in *imageName* to return where (*x1 ,y1*) is the top left and (*x2*,*y2*) is the bottom right (or bottom right corner if not specified). Includes the left and top edges but not the bottom or right edges.
-grayscale	Image data will be returned in grayscale format.
imageName **get***x y*	Returns a 3 element list representing the RGB color components of the pixel at (*x*,*y*) in *imageName*.
imageName **put***data ?option value ...?*	Inserts data from string *data* into *imageName* using given options.
-format *formatName*	(Tk 8.3+) Specify format for *data* (default is auto select). The **GIF**, **PGM**, **PPM** formats are supported.
-from *x1 y1 x2 y2*	(Tk 8.3+) Specifies rectangular sub-region (default is whole image) of the image in *data* to put into *imageName* where (*x1*,*y1*) is the top left and (*x2*,*y2*) is the bottom right (or bottom right corner if not specified). Includes the left and top edges but not the bottom or right edges.
-shrink	(Tk 8.3+) Will shrink image in *data* so it fits within the current bottom-right corner of *imageName* without affecting the **image create** settings for **-height** or **-width**.
-to *x y*	Specifies the top left corner (*x*,*y*) of the region (default is 0,0) within *imageName*, into which the pixels from the image in *data* will be put. In Tk versions up to 8.2.3, args are *x1 y1 x2 y2*.
imageName **read***fileName ?option value ...?*	Reads image data from *fileName* into *imageName* using given options.
-format *formatName*	Specify format for *fileName* (default is auto select). The **GIF**, **PGM**, **PPM** formats are supported.
-from *x1 y1 x2 y2*	Specifies rectangular sub-region (default is whole image) of the image in *fileName* to read into *imageName* where (*x1*,*y1*) is the top left and (*x2*,*y2*) is the bottom right (or bottom right corner if not specified). Includes the left and top edges but not the bottom or right edges.
-shrink	Will shrink image from *fileName* so it fits within the current bottom-right corner of *imageName* without affecting the **image create** settings for **-height** or **-width**.
-to *x y*	Specifies the top left corner (*x*,*y*) of the region (default is 0,0) within *imageName*, into which the pixels from the image in *fileName* will be put.

imageName **redither**	Redither the image. Used when multiple pieces are used for an image and dithering isn't exact.
imageName **transparency** *subcommand ?arg ...?*	(Tk 8.4+) Allows examination and manipulation of transparency info. *Subcommands* and *args* are:
get *x y*	Returns a boolean indicating if the pixel at (*x*,*y*) is transparent.
set *x y boolean*	If true, make the pixel at (*x*,*y*) transparent or opaque if false.
imageName **write***fileName ?option value ...?*	Writes image data from *imageName* into file *fileName*.
-background *color*	(Tk 8.3+) If specified, all transparent pixels will be replaced with *color*.
-format *formatName option*	Specify format for *fileName* (default is auto select). The supported *formatNames* are: **GIF87**, **GIF89**, **PGM**, or **PPM**.
-from *x1 y1 x2 y2*	Specifies rectangular sub-region (default is whole image) of the image in *imageName* to write to *fileName* where (*x1*,*y1*) is the top left and (*x2*,*y2*) is the bottom right (or bottom right corner if not specified). Includes the left and top edges but not the bottom or right edges.
-grayscale	(Tk 8.3+) Image data will be written in grayscale format.

3.13 Label Widget

Command	Description
label *pathName ?options?*	Creates a label widget *pathName* with *options* and returns the new widget's path name. When invoked, *pathName* must not exist, but *pathName*'s parent should. A label widget is used to display a text string, bitmap, or image. Multiple fonts within the text string are not supported.

Label Widget Options

Standard

See Common Options and Resources in Options and Resources for full details.

-activebackground (Tk 8.3.2+)	**-disabledforeground** (Tk 8.3.2+)	**-padx**
-activeforeground (Tk 8.3.2+)	**-font**	**-pady**
-anchor	**-foreground**	**-relief**
-background	**-highlightbackground**	**-takefocus**
-bitmap	**-highlightcolor**	**-text**
-borderwidth	**-highlightthickness**	**-textvariable**
-compound (8.4+)	**-image**	**-underline**
-cursor	**-justify**	**-wraplength**

Label Widget Specific

See Coordinates in Options and Resources for screen unit options.

Configure Option	Resource Name	Resource Class	Description
-height *height*	**height**	**Height**	Height of label widget (default is to auto size) in screen units (bitmap or image) or lines of text (text).
-state *state*	**state**	**State**	(Tk 8.3.2+) State of label widget. Options are: **active** (use **activeforeground** and **activebackground**), **disabled** (use **disabledforeground** and **background**), **normal** (use **foreground** and **background**).
-width *width*	**width**	**Width**	Width of label widget (default is to auto size) in screen units (bitmap or image) or characters (text).

Label Widget Commands

Command	Description
pathName **cget** *option*	Returns the current value of the configuration *option*. See Label Widget Options above for *options* .
pathName **configure** *?option? ?value? ?option value ...?*	Change the configuration *option* to*value*. Without *value*, a list describing *option* is returned. Without *option*, a list of all available options for *pathName* is returned. For multiple options an empty string is returned. See Label Wiget Options above for *options*.

3.14 Labelframe Widget

Command	Description
labelframe *pathName ?options?*	(Tk 8.4+) Creates a labelframe widget *pathName* with *options* and returns the new widget's path name. When invoked, *pathName* must not exist, but *pathName*'s parent should. A labelframe widget is used as a container for complex window layouts and has the featues of a frame plus the capability to display a label.

Labelframe Widget Options

Standard

See Common Options and Resources in Options and Resources for full details.

-borderwidth	**-highlightbackground**	**-pady**
-cursor	**-highlightcolor**	**-relief**
-font	**-highlightthickness**	**-takefocus**
-foreground	**-padx**	**-text**

LabelFrame Specific

See Coordinates in Options and Resources for screen unit options.

Configure Option	Resource Name	Resource Class	Description
-background *color*	**background**	**Background**	Same as standard **-background** expect if set to empty string, the widget will not display or allocate a colormap entry for the background or border color.
-class *name*	**class**	**Class**	Specifies class name to use in querying the option database and for bindings. Can not be changed with **configure** command.
-colormap *colormap*	**colormap**	**Colormap**	Specifies colormap (default is same as parent) to use for the window where *colormap* can be **new** (allocate new colormap) or the name of another window on same display with same visual. Can not be changed with **configure** command.
-container *boolean*	**container**	**Container**	Specifies whether the frame will be a container to embed another application. Can not be changed with **configure** command.
-height *height*	**height**	**Height**	Height of frame in screen units.
-labelanchor *anchorPos*	**labelAnchor**	**LabelAnchor**	Specifies where to position label in widget. Valid *anchorPos* values: **n**, **ne**,**en** ,**e**,**es**,**se**,**s**,**sw**,**ws**,**w**,**wn** , and **nw** (default).
-labelwidget *pathName*	**labelWidget**	**LabelWidget**	Widget to use as the label in the frame. Overrides **-text** option. Widget must already exist.
-visual *visual*	**visual**	**Visual**	Specifies the visual to use for the widget if different from parent. See Screen or Window Visuals in Toplevel for *visual* options. Can not be changed with **configure** command.
-width *width*	**width**	**Width**	Width of labelframe in screen units.

Labelframe Commands

Command	Description
pathName **cget** *option*	Returns the current value of the configuration *option*. See Labelframe Options above for *options* .
pathName **configure** *?option? ?value? ?option value ...?*	Change the configuration *option* to*value*. Without *value*, a list describing *option* is returned. Without *option*, a list of all available options for *pathName* is returned. For multiple options an empty list is returned. See Labelframe Options above for *options*.

3.15 Listbox Widget

Command	Description
listbox *pathName ?options?*	Creates a listbox widget *pathName* with *options* and returns the new widget's path name. When invoked, *pathName* must not exist, but *pathName*'s parent should. A listbox widget is used to display a list of strings, one per line. Listbox widgets are only one column.

Listbox Widget Options

Standard

See Common Options and Resources in Options and Resources for full details.

-background	**-foreground**	**-selectborderwidth**
-borderwidth	**-highlightbackground**	**-selectforeground**
-cursor	**-highlightcolor**	**-setgrid**
-disabledforeground (Tk 8.4+)	**-highlightthickness**	**-takefocus**
-exportselection	**-relief**	**-xscrollcommand**
-font	**-selectbackground**	**-yscrollcommand**

Listbox Widget Specific

Configure Option	Resource Name	Resource Class	Description
-activestyle *style*	**activeStyle**	**ActiveStyle**	(Tk 8.4+) Style in which to draw active element. *Style* options are: **dotbox** to show a focus ring around the active element, **none**, or **underline** (default) to underline the active element.
-height*height*	**height**	**Height**	Height of window (0 or default is to auto size) in lines of text.
-listvariable *var*	**listVariable**	**Variable**	(Tk 8.3+) Specifies name of variable *var* which contains a list to be displayed in the listbox. Automatically updates listbox when *var* is altered. Unsetting *var* while in use by the listbox, will be ignored.
-selectmode *mode*	**selectMode**	**SelectMode**	Specifies selection manipulation mode. *Mode* can be: **single** where only 1 element can be selected at a time, **browse** (default) where only 1 element can be selected or dragged at a time, **multiple** where multiple elements can be selected without affecting other selections, or **extended** where multiple elements can be selected but other selected elements become deselected.
-state *state*	**state**	**State**	(Tk 8.4+) State of label widget. Options are:**disabled** where items cannot be inserted or deleted (use **disabledforeground** and **background**) or **normal** (use **foreground** and **background**).
-width *width*	**width**	**Width**	Width of window (0 or default is to auto size) in characters (for proportional fonts, char size is for character "0").

Indicies or Character Positions

Some listbox commands support the use of an *index* to locate an element within the listbox or a character within a listbox element starting at 0. The following are the valid forms of specifying an *index* :

Index form	Description
number	A decimal number giving the index (starting from 0) of the element in the listbox. If *number* < 0, the 0 is used, if *number* > number of elements, then **end** is used.
active	Indicates the element that has the location cursor.
anchor	Selection anchor point as set by the **selection anchor** command.
end	Indicates the end of the listbox. Usually this is last element in the listbox, but for a few commands such as **index** and **insert** it refers to the element just after the last one.
@*x,y*	Indicates the element that covers coordinate *x,y* (pixel units) in the listbox window. If outside the window, it is set to the nearest legal value.

Listbox Widget Commands

For commands that use indicies, see Indicies or Character Positions above for options.

Command	Description

pathName **activate** *index*	Sets active element (for by keybaord bindings) in listbox to *index*. If outside the listbox range, it is set to the nearest element.
pathName **bbox***index*	Returns a list of four elements *x y w h* , giving an approximate bounding box of the text in element *index* . Coordinates *x,y* are top-left corner of text at *index* ,*w* is width of text, and *h* is height of text in pixels. Returns an empty string if element *index* is not visible on the screen or is invalid.
pathName **cget** *option*	Returns the current value of the configuration *option*. See Listbox Widget Options above for *options* .
pathName **configure** *?option? ?value? ?option value ...?*	Change the configuration *option* to *value*. Without *value*, a list describing *option* is returned. Without *option*, a list of all available options for *pathName* is returned. For multiple options an empty string is returned. See Listbox Widget Options above for *options*.
pathName **curselection**	Returns a list of numerical indices for all of the elements in the listbox that are currently selected or empty string if none.
pathName **delete** *first ?last?*	Deletes one or more elements in listbox from index *first* to index *last*. Without *last* only element at index *first* is deleted.
pathName **get** *first ?last?*	Returns a list of the contents of listbox elements from index *first* to index *last* (default is *first)*. Returns an empty string for invalid indicies.
pathName **index** *index*	Returns the numerical index corresponding to *index*.
pathName **insert** *index ?element ...?*	Insert zero or more elements just before the element at *index*. If *index* is **end**, new elements are added to the end of the list.
pathName **itemcget** *index option*	(Tk 8.3+) Returns the current value of the configuration *option* for the element at *index*. See Listbox Widget Options above for *options*.
pathName **itemconfigure** *index ?option? ?value? ?option value ...?*	(Tk 8.3+) Change the configuration *option* for element at *index* to *value*. Without *value*, a list describing *option* is returned. Without *option*, a list of all available options is returned. For multiple options an empty list is returned. Supported *options* are: **-background**, **-foreground**, **-selectbackground**, and **-selectforeground**.
pathName **nearest** *y*	Returns the index of the visible element nearest coordinate *y*.
pathName **scan** *option args*	Implements scanning on listbox widgets. *Options* are:
mark *x y*	Records *x, y,* and the current view in the listbox window. Typically associated with mouse button press in widget.
dragto *x y*	Adjusts the view by 10 times the difference between the coordinate *x,y* and the last **mark** *x ,y* coordinate. Used with mouse motion events to produce high speed dragging.
pathName **see** *index*	Adjust the view in the listbox so that the element at *index* is visible in the center of the listbox. If the element is near the beginning or end then the element will be visible at the edge.
pathName **selection** *option arg*	Manipulates the selection within a listbox based on *option*. Vaild *options* and *args* are:
anchor *index*	Sets the selection anchor to the element at *index*. If *index* is invalid, the element closest to *index* will be used.
clear *first ?last?*	Clears from the selection the elements between indicies *first* and *last* (default is *first*), inclusive, without affecting the selection state of elements outside the range.
includes *index*	Returns 1 if the element at *index* is currently selected, 0 if not.
set *first ?last?*	Selects all of the elements between indicies *first* and *last* (default is *first)*, inclusive, without affecting the selection state of elements outside the range.
*pathName***size**	Returns the total number of elements in the listbox.
pathName **xview** ?*option args*?	Query or change the horizontal listbox view. Without any *options*, returns a two element list specifying the start and end of the visible fraction (from 0 to 1) of the horizontal span of the widget between the left and right edges of the window. Vaild *options* and *args* are:
index	Adjust window view to display the character at position *index* at the left edge of window.
moveto *fraction*	Adjust window view so that *fraction* (from 0 to 1) of the total width of the listbox widget is off-screen to the left.
scroll *number* **pages**	Shift the view left (*number* < 0) or right (*number* > 0) by *number* screenfuls.
scroll *number* **units**	Shift the view left (*number* < 0) or right (*number* > 0) by *number* average-width characters (proportional uses "0").

pathName **yview** ?*option args*?	Query or change the vertical listbox view. Without any *options*, returns a two element list specifying the start (element at top of window) and end (element just after element at the bottom of window) of the visible fraction (from 0 to 1) of the vertical span of the widget between the top and bottom edges of the window. Vaild *options* and *args* are:
index	Adjust window view to display the element at *index* at the top edge of window.
moveto *fraction*	Adjust the view in the window so that element at *fraction* (from 0 to 1) is at the top edge of the window.
scroll *number* **pages**	Shift the view up (*number* < 0) or down (*number* > 0) by *number* screenfuls.
scroll *number* **units**	Shift the view up (*number* < 0) or down (*number* > 0) by *number* lines.

Default Listbox Widget Bindings

Event	Description
<Button-1>	Make the element under the mouse pointer the active element.
<B1-Motion>	If the selection mode is **extended**, extend the selection.
<Shift-1>	If the selection mode is **extended**, modifies the selection to consist of the elements between the anchor and the element under the mouse pointer, inclusive.
<Control-1>	If the selection mode is **extended**, set anchor to element under the mouse pointer and toggle its selection state. The selection state of other elements isn't changed.
<Control-B1-Motion>	If the selection mode is **extended**, the selection state of all elements between the anchor and the element under the mouse is set to match that of the anchor element; the selection state of all other elements remains what it was before the toggle operation began.
<B1-Leave>	Adjusts view in listbox in direction of mouse pointer more quickly.
<B1-Enter>	Stops quick adjustment of view in listbox in direction of mouse pointer.
<ButtonRelease-1>	Stops quick adjustment of view in listbox in direction of mouse pointer and activates current listbox entry.
<Double-1>	No function.
<Button-2>	Mark entry for start of scanning.
<B2-Motion>	Drag the contents of the listbox at high speed in the direction the mouse moves.
<Up>	Move the location cursor (active element) up by one element. If the selection mode is **browse** or **extended** then the new active element is also selected and all other elements are deselected. In **extended** mode the new active element becomes the selection anchor.
<Down>	Move the location cursor (active element) down by one element. If the selection mode is **browse** or **extended** then the new active element is also selected and all other elements are deselected. In **extended** mode the new active element becomes the selection anchor.
<Shift-Up>	In **extended** mode, move the location cursor (active element) up one element and also extend the selection to that element.
<Shift-Down>	In **extended** mode, move the location cursor (active element) down one element and also extend the selection to that element.
<Left>	Scroll the listbox view left by the width of the character **0**.
<Right>	Scroll the listbox view right by the width of the character **0**.
<Control-Left>	Scroll the listbox view left by the width of the window.
<Control-Right>	Scroll the listbox view right by the width of the window.
<Prior>	Move the location cursor (active element) and the listbox view up by one page (the height of the window).
<Next>	Move the location cursor (active element) and the listbox view down by one page (the height of the window).
<Control-Prior>	Scroll the listbox view up by one page (the height of the window).
<Control-Next>	Scroll the listbox view down by one page (the height of the window).
<Home>	Scroll the listbox horizontally to the left edge.
<End>	Scroll the listbox horizontally to the right edge.
<Control-Home>	Sets the location cursor to the the first element in the listbox, selects that element, and deselects everything else in the listbox.

<Control-End>	Sets the location cursor to the the last element in the listbox, selects that element, and deselects everything else in the listbox.
<Shift-Control-Home>	In **extended** mode, extends the selection to the first element in the listbox. In **multiple** mode, moves the location cursor to the first element in the listbox.
<Shift-Control-End>	In **extended** mode, extends the selection to the last element. In **multiple** mode, moves the location cursor to the last element.
<space>	Select the element at the location cursor and make it the active element.
<Select>	Select the element at the location cursor and make it the active element.
<Shift-Control-space>	In **extended** mode, extend the selection from the anchor to the active element.
<Shift-Select>	In **extended** mode, extend the selection from the anchor to the active element.
<Escape>	In **extended** mode, cancels the most recent selection and restores all the elements in the selected range to their previous selection state
<Control-slash>	In **browse** or **extended** modes, selects everything in the widget. In **single** and **browse** modes, selects the active element and deselects everything else.
<Control-backslash>	In **extended**, **multiple** , and **single** modes, deselects everything in the widget.
<MouseWheel>	(MS Windows only) Scroll listbox vertically by several entries in direction of wheel scroll.
<Button-4> or**<Button-5>**	(Unix only) Equivalent of **<MouseWheel>** to scroll listbox up (**<4>**) or down (**<5>**) by several entries.
<<Copy>>	Copies the selection in the widget to the clipboard, if there is a selection.
<<ListboxSelect>>	(Tk 8.3+) Virtual event is generated whenever the selection in a listbox changes.

3.16 Menu Widget

Command	Description
menu *pathName ?options?*	Creates a top-level menu widget *pathName* with *options* and returns the new widget's path name. When invoked, *pathName* must not exist, but *pathName*'s parent should. A menu widget is used to display a collection of one-line entries arranged in one or more columns. The several types of entries can be combined in a single menu. The entire menu is one widget. There are three types of menus: **menubar**, **normal**, and **tearoff**. For **menubar** and torn-off menus, a clone of the original menu is made. This clone is a menu widget in its own right, but it is a child of the original. Changes in the configuration of the original are reflected in the clone. Clones are destroyed when either the tearoff or menubar is closed, or when the original menu is destroyed.
tk_menuSetFocus *pathName*	Used by several of the menu bindings to save the current focus and set the focus to the menu widget *pathName* .
tk_popup *menu x y ?entry?*	Posts a pop-up menu *menu* with the entry at index *entry* (default is menu's upper left corner) positioned at root coordinate *x,y*

Menu Widget Options

Standard

See Common Options and Resources in Options and Resources for full details.

-activebackground	**-borderwidth**	**-foreground**
-activeborderwidth	**-cursor**	**-relief**
-activeforeground	**-disabledforeground**	**-takefocus**
-background	**-font**	

Menu Widget Specific

Configure Option	Resource Name	Resource Class	Description
-postcommand *tclCommand*	**postCommand**	**Command**	Specify Tcl command to invoke immediately before the menu is posted. Returns result.
-selectcolor *color*	**selectColor**	**Background**	Specifies indicator color for checkbutton and radiobutton entries.
-tearoff *boolean*	**tearOff**	**TearOff**	Specifies whether to include a tear-off entry at top of menu as entry 0 or not (default).
-tearoffcommand *tclCommand*	**tearOffCommand**	**TearOffCommand**	Specifies Tcl command to be invoked when the menu is torn off. The command excuted is: {*tclCommandmenu pathName* } {*torn-off menu pathName*}.
-title *string*	**title**	**Title**	Use *string* as the window title of the torn-off menu. If set to the empty string, the menubutton title or the cascade item text will be used.
-type *type*	**type**	**Type**	Specifies menu *type* at creation. *Type* can be: **menubar** (Set menu to be toplevel window menubar), **tearoff** (A tear-off entry of dashed lines appears at the top of the menu if enabled. When selected, creates a copy of the menu and submenus as a torn-off menu in a new window), or **normal** (normal cascade menu for either a top or lower level window).

Indicies

Some menu commands support the use of an *index* to locate an entry within the menu starting at 0. The following are the valid forms of specifying an *index*:

Index form	Description
number	A decimal number giving the entry (starting from 0) in the menu.
active	Indicates the entry that is currently active. If no entry is active, then this form is equivalent to **none** .
end	Entry at the bottom of the menu. If no entries, then this form is equivalent to **none**.
last	Same as **end**.
none	Indicate "no entry at all" and can be used with **activate** option to deactivate all entries in a widget.
@*y*	Indicates the entry closest to*y*-coordinate (pixel units) in the menu.
pattern	*Pattern* is pattern-matched using Pattern Globbing against the label of each entry in the menu, in order from the top down, until a matching entry is found.

Menu Widget Commands

For commands that use indicies, see Indicies above for options.

Command	Description
pathName **activate** *index*	Change state of entry at *index* to be the sole active entry in menu.
pathName **add** *type ?option value ...?*	Add new entry of type *type* to bottom of menu. See Entry Types below for *types*. See Menu Entries below for *options*.
pathName **cget***option*	Returns the current value of the configuration *option*. See Menu Widget Options above for *options* .
pathName **clone** *newPathName ?cloneType?*	Makes a clone of menu as a new menu*newPathName* of type*cloneType* (see **-type**).
pathName **configure***?option? ?value? ?option value ...?*	Change the configuration *option* to*value*. Without *value*, a list describing *option* is returned. Without *option*, a list of all available options for *pathName* is returned. For multiple options an empty string is returned. See Menu Widget Options above for *options*.
pathName **delete** *index1 ?index2?*	Delete all entries between *index1* and *index2* (default is *index1*) inclusive. Can't delete tear-off entries.
pathName **entrycget** *index option*	Return current value of *option* for entry at *index*. See Menu Entries below for *options*.
pathName **entryconfigure** *index ?option value ...?*	Change the configuration *option* to*value* for entry at *index*. Without *value*, a list describing *option* is returned. Without *option*, a list of all available options for the entry is returned. See Menu Entries below for *options*.
pathName **index** *index*	Returns the numerical index corresponding to*index* or none for index **none**.
pathName **insert** *index type ?option value ...?*	Insert new entry of type *type* to menu just before the entry at *index*. Entries can not be inserted before the tearoff entry if used. See Entry Types below for *types*. See Menu Entries below for *options*.
pathName **invoke** *index*	Invoke the action of the menu entry at *index* .
pathName **post** *x y*	Post or display menu *pathName* at root-window coordinates *x,y* (should be upper right corner of entry). The coordinates are adjusted if necessary to guarantee that the entire menu is visible on the screen.
pathName **postcascade** *index*	Post submenu associated with cascade entry at *index* and unpost any previously posted menu.
pathName **type** *index*	Returns type of entry at *index*. SeeEntry Types below for types.
pathName **unpost**	(Tk 8.4+) On Unix, unpost or unmap *pathName* so it is no longer displayed. This is handled automatically on Windows and Mac.
pathName **yposition** *index*	Returns the y-coordinate within the menu window of the topmost pixel in the entry specified by *index*.

Entry Types

Entry Type:	Description:
cascade	Menu entry with an associated submenu specified by **-menu** option which allows the construction of cascading menus. Submenus are posted and unposted via the **postcascade** command. Except on Windows, the **-command** option is evaluated each time the entry is invoked.
checkbutton	Behaves like a checkbutton widget where if invoked it toggles between selected and deselected states. Sets the global variable specified by **-variable** to the **-onvalue** value when selected and to the **-offvalue** value when deselected. The "on" indicator color is set by the **-selectcolor** option and the **-command** option is evaluated each time the entry's state is toggled.
command	Behaves like a button widget and when invoked, the **-command** option is evaluated.
radiobutton	Behaves like a radiobutton widget where only one entry within a group may be selected at a time. When an entry is selected, the **-value** value is stored to the global variable specified by **-variable** and previously specified entry is unselected. The "selected" indicator color is set by the **-selectcolor** option and the **-command** option is evaluated each time the entry's state becomes selected.
separator	Displays a horizontal dividing line.

Menu Entries

General

The following options are valid for cascade, checkbutton, command, and radiobutton entries. They are not valid for separator and tear-off entries.

-activebackground	**-bitmap**	**-foreground**
-activeforeground	**-compound** (8.4+)	**-image**
-background	**-font**	**-underline**

Menu Entry Specific

The following options work for all cascade, checkbutton, command, and radiobutton entries unless otherwise specified. See Common Options and Resources in Options and Resources for full details.

Option	Description
-accelerator *string*	Specifies string to display at right side of menu entry. Used for accelerator keystroke sequence to invoke entry. Not valid for separator and tear-off entries.
-columnbreak *value*	When *value* is set to 1, entry appears at top of a new column in menu. Default *value* of zero puts entry under previous entry.
-command *tclCommand*	Tcl command to evaluate when entry is invoked. Not valid for separator and tear-off entries.
-hidemargin *value*	Specifies whether the standard margins are drawn (default value of 0) or not (value of 1) around the entry.
-indicatoron *boolean*	Specifies whether the checkbutton or radiobutton entry indictor should be displayed (default) or not.
-label *string*	Text string to display in the menu entry. Not valid for separator and tear-off entries.
-menu *pathName*	Specifies pathname of submenu to post when cascade entry is active.
-offvalue *value*	Value to store in checkbutton entry's associated variable when deselected.
-onvalue *value*	Value to store in checkbutton entry's associated variable when selected.
-selectcolor *color*	Specifies indicator's color for checkbutton and radiobutton entries. The default value of empty string sets the color to the **-selectcolor** option for the menu.
-selectimage *image*	Specifies image to show instead of **-image** when checkbutton and radiobutton entries are selected.
-state *state*	Specifies state of entry. Options are: **active** (use **activeforeground** and **activebackground**),**disabled** (use **disabledforeground** and **background**), **normal** (use **foreground** and **background**). Not valid for separator entries.
-value *value*	Value to store in radiobutton entry's associated variable when selected.
-variable *variable*	Specifies name of the global variable to set when the checkbutton or radiobutton is selected and deselected for checkbutton.

Menu Entry Format:

Field Name:	Description:
Main field	The main field is a label in the form of a text string, a bitmap, or an image, controlled by the **-label**,**-bitmap**, and **-image** options for the entry.
Accelerator	If the **-accelerator** option is specified for an entry, a second textual field is displayed to the right of the label. It describes a keystroke sequence that may be used to invoke the same result as the menu entry.
Indicator	The indicator is displayed to the left of the entry's string for only checkbutton or radiobutton entries. It indicates whether the entry is selected or not.

Special System Dependent Menus:

Platform	Menu Name	Description
Mac	***.menuName*.apple**	Special Apple menu (Apple logo) that appears first on menubar. Adds user's Apple Menu Items folder to bottom of menu.
Mac	***.menuName*.help**	Special right-justified Help menu. Adds Apple help items to top of menu.
Windows	***.menuName*.system**	Windows System menu. Adds Microsoft items to top of menu.
Unix	***.menuName*.help**	Special right-justified Help menu.

Menu Configurations

Config	Decsription
Pulldown Menus in Menubar	Menu widget with multiple cascade entries and associated pull down menus. Add to toplevel window using the **-menu** option.
Pulldown Menus in Menu Buttons	Menubutton widget with multiple top-level menus arranged in a row within a menubar window. Each top-level menu can be cascade with associated submenus. The top-level menu must be a child of the menubutton, and each submenu must be a child of the menu that refers to it.
Popup Menus	Posts the top-level menu via **tk_popup** in response to a mouse button press or keystroke.
Option Menus	Created with **tk_optionMenu** and consists of a menubutton with an associated menu that allows you to select one of several values. The current value is displayed in the menubutton and is also stored in a global variable.
Torn-off Menus	Created by invoking the tear-off entry at the top of an existing menu. The default bindings will create a new menu that is a copy of the original menu and leave it permanently posted as a top-level window. The torn-off menu behaves just the same as the original menu.

Menu Widget Bindings

Event	Description
<Enter>	Entry underneath the mouse cursor activates.
<Leave>	All of the entries in the menu deactivate, except in the special case where the mouse moves from a menu to a cascaded submenu.
<FocusIn>	none
<Motion>	Active entry changes to track the mouse.
<ButtonPress>	Change the posted cascade entry (if any) to match the mouse position,
<ButtonRelease>	Active entry is invoked and if a menu, unpost it unless it is a tear-off.
<space>	Invoke the active entry and unpost the menu.
<Return>	Invoke the active entry and unpost the menu.
<Escape>	Aborts a menu selection in progress without invoking any entry. It also unposts the menu unless it is a torn-off menu.
<Left>	Moves to the next menu on the left. For cascade submenus, the submenu is also unposted and the current menu entry becomes the cascade entry in the parent. For top-level menus posted from a menubutton, then the current menubutton is unposted and the next menubutton to the left is posted, otherwise the key has no effect. The left-right order of menubuttons is determined by their stacking order with the lowest menubutton on the left.
<Right>	Moves to the next menu on the right. For cascade entries, the submenu is also posted and the current menu entry becomes the first entry in the submenu, otherwise if the current menu was posted from a menubutton, then the current menubutton is unposted and the next menubutton to the right is posted.
<Up>	Activate the next higher entry in the menu. When the top of the menu is reached, the active entry wraps around to the bottom.
<Down>	Activate the next lower entry in the menu. When the bottom of the menu is reached, the active entry wraps around to the top.
<KeyPress>	If any of the entries in a menu have letters underlined with with **-underline** option, then pressing one of the underlined letters (or its upper-case or lower-case equivalent) invokes that entry and unposts the menu.
<Alt-KeyPress>	Implements keyboard traversal of menus. Given an ASCII character "char", it looks for a menubutton with that character underlined. If one is found, it posts the menubutton's menu.
<F10>	Traverses to the first menubutton in the toplevel for a given window, and posts that menubutton's menu.
<<MenuSelect>>	Virtual event generated whenever a menu's active entry is changed.

3.17 Menubutton Widget

Command	Description
menubutton *pathName ?options?*	Creates a menubutton widget *pathName* with *options* and returns the new widget's path name. When invoked,*pathName* must not exist, but *pathName*'s parent should. A menubutton widget is used to display a textual string, bitmap, or image and is associated with a menu widget. Selecting the menu button displays the associated menu. Text can only use a single font.
tk_optionMenu *pathNamevarName value ?value ...?*	Creates a menubutton with name *pathName* and an associated menu of options. When the menubutton is selected, the associated option menu pops up and the user can select from the *value* args. The selected value is stored in *varName* and displayed as the label in the menubutton. Returns the pathname of the menu associated with *pathName*.

Menubutton Widget Options

Standard

See Common Options and Resources in Options and Resources for full details.

-activebackground	**-disabledforeground**	**-padx**
-activeforeground	**-font**	**-pady**
-anchor	**-foreground**	**-relief**
-background	**-highlightbackground**	**-takefocus**
-bitmap	**-highlightcolor**	**-text**
-borderwidth	**-highlightthickness**	**-textvariable**
-compound (8.4+)	**-image**	**-underline**
-cursor	**-justify**	**-wraplength**

Menu Widget Specific

Configure Option	Resource Name	Resource Class	Description
-direction *direction*	**direction**	**Height**	Specifies where the menu will popup. *Direction* can be **above**, **below** (default), **left**, **right**, or **flush** (over) with the menubutton.
-height *height*	**height**	**Height**	Height of menubutton widget (default is to auto size) in screen units (bitmap or image) or lines of text (text). See Coordinates in Options and Resources for screen unit options.
-indicatoron *boolean*	**indicatorOn**	**IndicatorOn**	Specifies whether an indictor should be displayed to the right of the menubutton (default) and treated as a option menubutton.
-menu *pathName*	**menu**	**MenuName**	Specifies *pathName* of menu widget to post when button is invoked. Menu must be a child of the menubutton.
-state *state*	**state**	**State**	Specifies state of entry. Options are: **active** (use **activeforeground** and **activebackground**), **disabled** (use **disabledforeground** and **background**),**normal** (use **foreground** and **background**).
-width *width*	**width**	**Width**	Width of menubutton widget (default is to auto size) in screen units (bitmap or image) or characters (text). See Coordinates in Options and Resources for screen unit options.

Menubutton Widget Commands

Command	Description
pathName **cget***option*	Returns the current value of the configuration *option*. See Menubutton Widget Options above for *options* .
pathName **configure***?option? ?value? ?option value ...?*	Change the configuration *option* to*value*. Without *value*, a list describing *option* is returned. Without *option*, a list of all available options for *pathName* is returned. For multiple options an empty string is returned. See Menubutton Widget Options above for *options*.

Menubutton Widget Bindings

Event	Description
<Enter>	Menubutton is activated.
<Leave>	Menubutton is deactivated and returns to its normal state.
<Button-1>	Post menu for menubutton or activate entry.
<Motion>	Deactivates previous entry and activates new entry.
<B1-Motion>	Deactivates previous entry and activates or posts submenus for new entry.
<ButtonRelease-1>	If the release happens inside the menubutton then leave its menu posted with element 0 activated, otherwise unpost the menu without invoking any menu entry. If menu entry is active, invoke entry and unpost menu.
<Alt-KeyPress>	Implements keyboard traversal of menus. Given an ASCII character "char", it looks for a menubutton with that character underlined. If one is found, it posts the menubutton's menu.
<F10>	Traverses to the first menubutton in the toplevel for a given window, and posts that menubutton's menu.
<space>	Invoke the active entry and unpost the menu.
<Return>	Invoke the active entry and unpost the menu.
<<MenuSelect>>	Virtual event generated whenever a menu's active entry is changed.

3.18 Message Widget

Command	Description
message *pathName ?options?*	Creates a message widget *pathName* with *options* and returns the new widget's path name. When invoked, *pathName* must not exist, but *pathName*'s parent should. A message widget is used to display a textual string. Text can only use a single font and is broken up on word boundaries if possible. Tab characters are replaced with blank space up to the next 8-character boundary, newlines cause line breaks, and control characters and other undefined characters in the font are displayed as a 8-bit hex backslash sequence (**x***hh*).

Message Widget Options

Standard

See Common Options and Resources in Options and Resources for full details.

-anchor	**-highlightbackground**	**-relief**
-background	**-highlightcolor**	**-takefocus**
-borderwidth	**-highlightthickness**	**-text**
-cursor	**-justify**	**-textvariable**
-font	**-padx**	
-foreground	**-pady**	

Message Widget Specific

Configure Option	Resource Name	Resource Class	Description
-aspect *integer*	**aspect**	**Aspect**	Ratio of text width to text height for text display. Formula is: ratio = 100*width/height. 100 = text is as wide as it is tall. 150 (default) = text is 1.5 times wide as it is tall.
-width *width*	**width**	**Width**	Width of menubutton widget (default is to auto size) in screen units (bitmap or image) or characters (text). See Coordinates in Options and Resources for screen unit options.

Message Widget Commands

Command	Description
pathName **cget***option*	Returns the current value of the configuration *option*. See Message Widget Options above for *options* .
pathName **configure***?option? ?value? ?option value ...?*	Change the configuration *option* to*value*. Without *value*, a list describing *option* is returned. Without *option*, a list of all available options for *pathName* is returned. For multiple options an empty string is returned. See Message Widget Options above for *options*.

3.19 Options and Resources

A widget is a term used in describe a component of a graphical user interface (GUI). In general, a widget name (*pathName)* is the concatenation of its parent's name followed by a period (unless the parent is the root window ".") and a string containing no periods (eg. .mainframe.buttonframe.b1).

Command	Description
option add *pattern value ?priority?*	Adds new option with name or class *pattern* set to *value* at *priority* (0-100) to the database. Priority level symbols are: **widgetDefault** (level 20 - Use default values hard-coded into widgets), **startupFile** (level 40 - Use options in app-specific startup files), **userDefault** (level 60 - User specific options from .Xdefaults, X server database, user specific start-up files), and **interactive** (level 80 - default - Options specified interactively after the application starts running).
option clear	Clears option database and reloads from user's Xdefaults on next add or get.
option get *window name class*	Returns the option value with highest priority level for *window* under *name* and *class* or empty string if none.
option readfile *fileName ?priority?*	Reads options from Xdefaults-style file into option database at *priority* (default is **interactive**).
tk_bisque	Set default color palette to old bisque (light brown) scheme.
tk_setPalette *color*	Changes the color scheme for Tk so the default background color is *color* and other default colors are computed using reasonable defaults.
tk_setPalette *option color ?option color ...?*	Set the default color for *option* in the color scheme. Modifies existing widgets using default values and adds to option database at priority **widgetDefault**. Must include **background** option in all cases. Available options are: **activeBackground**,**activeForeground**, **background**, **disabledForeground**, **foreground**,**highlightBackground**, **highlightColor**, **insertBackground**,**selectColor**, **selectBackground**, **selectForeground**, and **troughColor**.

Widget Options and Resources:

When a widget is created, the order for determining which configuration options to use is: command line options, resource database entries (name or class), then the hard coded value from widget implemenation. Options and resources are configured by:

Method	Syntax
Configure Option	*pathName* **configure** *option value* ?option value ...?
Resource Name	**option add** {app name or *}.{widget name}.{Resource name} *value*
Resource Class	**option add** Tk.{widget name}.{Resource Class} *value*

Common Options and Resources

The following is the list of common options and resources used by most widgets. In each widget section, the applicable options from the following list will be listed. See Coordinates below for screen unit options. See Colors below for *color* options.

Configure Option	Resource Name	Resource Class	Description
-activebackground *color*	**activeBackground**	**Foreground**	Background color of widget when it is active. Normally ignored when **tk_strictMotif** is set.
-activeborderwidth *width*	**activeBorderWidth**	**BorderWidth**	Border width of widget in screen units when it is active.
-activeforeground *color*	**activeForeground**	**Background**	Foreground color of widget when it is active.
-anchor *anchorPos*	**anchor**	**Anchor**	Where to position information in widget. Valid *anchorPos* values: **n**, **ne**, **e**,**se**,**s**,**sw**,**w**,**nw**, and **center**.
-background *color*	**background**	**Background**	Normal background color of widget (Also **-bg**).
-bitmap *bitmap*	**bitmap**	**Bitmap**	Bitmap to display in widget. SeeDefault Bitmaps below. Overrides **text** options. Set to empty string to re-enable text display. Options are:
	name	To use an existing bitmap *name*	
	@*fileName*	To load bitmap from *fileName*	
-borderwidth *width*	**borderWidth**	**BorderWidth**	Normal 3-D border width of widget in screen units. (Also **-bd**).
-compound *value*	compound	Compound	(Tk 8.4+) Specifies if the widget should display both an image and text, and if so, where the image should be placed relative to the text. Options are: **bottom**,**center**, **left**, **none** (default whiches uses **-image** and **-bitmap** options), **right**, and **top**.
-cursor *cursor*	**cursor**	**Cursor**	Cursor to display when mouse pointer is in widget. Valid *cursors*:
	name [fgColor [bgColor]]		Name of cursor (See Cursors). Optionally specify the foreground (default is black) and background (default is transparent) colors.
	@sourceName maskName fgColor bgColor		Get source and mask bits from files *sourceName* and *maskName*.
	@sourceName fgColor		Get source bits from file *sourceName* with transparent background. (Unix only)
	@sourceName		(Tk 8.3+) Load system cursor (.ani or .cur) from *sourceName*. (MS Windows only)

-disabledforeground *color*	**disabledForeground**	**DisabledForeground**	Foreground color of widget when it is disabled. If set to an empty string, the normal **foreground** color with stippled fill pattern is used.
-exportselection *boolean*	**exportSelection**	**ExportSelection**	Whether a selection in the widget should also be the X selection.
-font *font*	**font**	**Font**	Font to use when drawing text inside the widget. See Fonts.
-foreground *color*	**foreground**	**Foreground**	Normal foreground color of widget. (Also **-fg**).
-highlightbackground *color*	**highlightBackground**	**HighlightBackground**	Color of rectangle drawn around widget when it does not have the input focus.
-highlightcolor *color*	**highlightColor**	**HighlightColor**	Color of rectangle drawn around widget when it has the input focus.
-highlightthickness *width*	**highlightThickness**	**HighlightThickness**	Width of highlight rectangle drawn around widget when it has the input focus in screen units.
-image *imageName*	**image**	**image**	Image to display in the widget. Overrides **bitmap**. Set to empty string to re-enable bitmap or text display.
-insertbackground *color*	**insertBackground**	**Foreground**	Background color of area covered by the insertion cursor. Overrides **background** or **selectbackground** .
-insertborderwidth *width*	**insertBorderWidth**	**BorderWidth**	3-D border width to draw around the insertion cursor in screen units.
-insertofftime *milliseconds*	**insertOffTime**	**OffTime**	Time the insertion cursor should remain "off" in each blink cycle.
-insertontime *milliseconds*	**insertOnTime**	**OnTime**	Time the insertion cursor should remain "on" in each blink cycle.
-insertwidth *width*	**insertWidth**	**InsertWidth**	Insertion cursor width in screen units.
-jump *boolean*	**jump**	**Jump**	When scrollbars and scales connected to the widget notify widget of updates. **True** is delay until mouse button is released. **False** is continuously.
-justify *option*	**justify**	**Justify**	How to justify lines of text. Options are: **left** (default), **center**, or **right**.
-orient *option*	**orient**	**Orient**	Orientation the widget should for its layout. Options are: **horizontal**, **vertical**
-padx *width*	**padX**	**Pad**	Extra space in screen units to request for the widget in X-direction.
-pady *height*	**padY**	**Pad**	Extra space in screen units to request for the widget in Y-direction.
-relief *option*	**relief**	**Relief**	Desired widget border 3-D effect. Options are: **flat**, **groove**, **raised**, **ridge**,**solid**, or **sunken**.
-repeatdelay *milliseconds*	**repeatDelay**	**RepeatDelay**	Time a button or key must be held down before it begins to auto-repeat.
-repeatinterval *milliseconds*	**repeatInterval**	**RepeatInterval**	Time between auto-repeats once action has begun.
-selectbackground *color*	**selectBackground**	**Foreground**	Background color for selected items.
-selectborderwidth *width*	**selectBorderWidth**	**BorderWidth**	Width of border to draw around selected items in screen units.

-selectforeground *color*	**selectForeground**	**Background**	Foreground color for selected items.
-setgrid *boolean*	**setGrid**	**SetGrid**	Whether this widget controls the resizing grid for its toplevel window.
-takefocus *focusType*	**takeFocus**	**TakeFocus**	Determines if window accepts the focus during keyboard traversal. Options are:
	0	skip window	
	1	allow if viewable	
	empty string	Tk decides (skip if disabled, no key bindings, or not viewable)	
	other	evaluates as a Tcl script with window name lappended as an arg. Script must return 0, 1, or empty string.	
-text *string*	**text**	**Text**	Text string to be displayed inside the widget. Can include \n.
-textvariable *variable*	**textVariable**	**Variable**	Variable which contains a text string to be displayed inside the widget.
-troughcolor *color*	**troughColor**	**Background**	Trough color for scrollbar and scale widgets.
-underline *index*	**underline**	**Underline**	Integer index of a character to underline in the widget for keyboard traversal.
-wraplength *length*	**wrapLength**	**WrapLength**	Maximum line length for word-wrapping in screen units. 0 = no wrapping except for \n.
-xscrollcommand *cmdPrefix*	**xScrollCommand**	**ScrollCommand**	Prefix for a command used to communicate with horizontal scrollbars. Widget will execute cmd with args of *start* and *end* of current view. Values can be from 0 to 1.
-yscrollcommand *cmdPrefix*	**yScrollCommand**	**ScrollCommand**	Prefix for a command used to communicate with vertical scrollbars.

Default Bitmaps

All Platforms:		Mac only:			
error	**hourglass**	**accessory**	**edition**	**pfolder**	**trash**
gray12	**info**	**application**	**floppy**	**querydoc**	
gray25	**questhead**	**caution**	**folder**	**ramdisk**	
gray50	**question**	**cdrom**	**note**	**stationary**	
gray75	**warning**	**document**	**preferences**	**stop**	

Colors:

For color options the following are the valid options where *colorname* is a text string matching a color in the X server database and # starts a numeric specification of the red, green, and blue intensities. Each R, G, and B represents a hex digit. The four forms permit colors to be specified with 4-bit, 8-bit, 12-bit, or 16-bit values. When fewer than 16 bits are provided for each color, they represent the most significant bits of the color.

colorname	*#RGB*	*#RRGGBB*	*#RRRGGGBBB*	*#RRRRGGGGBBBB*

Commonly used colors for *colorname*:

DarkRed	**maroon**	**red**		**DeepPink**	**coral**	**pink**
gold	**goldenrod**	**yellow**	**LightYellow**	**DarkOrange**	**orange**	
brown	**chocolate**	**tan**	**wheat**	**chartreuse**		
DarkGreen	**green**	**honeydew**	**PaleGreen**	**aquamarine**	**turquoise**	
DarkBlue	**MidnightBlue**	**blue**	**LightBlue**	**DarkCyan**	**cyan**	**SkyBlue**
SlateBlue	**DodgerBlue**	**SteelBlue**	**CadetBlue**	**DarkViolet**	**purple**	**violet**
black	**DarkGray**	**gray**	**LightGray**	**bisque**	**white**	
gray1 to **gray100** or **grey1** to **grey100**				**OrangeRed**	**magenta**	**BlueViolet**

Special MS Windows Colors:

Color Name	Purpose
SystemButtonFace	Default background
SystemButtonText	Default foreground
SystemButtonHighlight	
SystemButtonShadow	
SystemHighlight	Default highlight background
SystemHighlightText	Default highlight foreground
SystemWindow	Default entry, list, text, etc. background
SystemWindowText	Default entry, list, text, etc. foreground
SystemWindowFrame	Default window frame color

Coordinates:

For options that take screen units, default value is in pixels unless one of the following optional suffix modifiers is present. Units can be floating point numbers. Coordinate (0,0) is in the top left corner of the widget. X-Windows and MS Windows 9x and ME use 16 bit coordinates. MS Windows NT and later use 32 bit coordinates.

c (centimeters)	**i** (inches)	**m**m (millimeters)	**p** (points where 1p = 1/72 inch)

Cursors:

Default cursors from /usr/include/X11/cursorfont.h on Unix:

arrow	**crosshair**	**iron_cross**	**right_tee**	**tcross**
based_arrow_down	**diamond_cross**	**left_ptr**	**rightbutton**	**top_left_arrow**
based_arrow_up	**dot**	**left_side**	**rtl_logo**	**top_left_corner**
boat	**dotbox**	**left_tee**	**sailboat**	**top_right_corner**
bogosity	**double_arrow**	**leftbutton**	**sb_down_arrow**	**top_side**
bottom_left_corner	**draft_large**	**ll_angle**	**sb_h_double_arrow**	**top_tee**
bottom_right_corner	**draft_small**	**lr_angle**	**sb_left_arrow**	**trek**
bottom_side	**draped_box**	**man**	**sb_right_arrow**	**ul_angle**
bottom_tee	**exchange**	**middlebutton**	**sb_up_arrow**	**umbrella**
box_spiral	**fleur**	**mouse**	**sb_v_double_arrow**	**ur_angle**
center_ptr	**gobbler**	**pencil**	**shuttle**	**watch**
circle	**gumby**	**pirate**	**sizing**	**xterm**
clock	**hand1**	**plus**	**spider**	**X_cursor**
coffee_mug	**hand2**	**question_arrow**	**spraycan**	
cross	**heart**	**right_ptr**	**star**	
cross_reverse	**icon**	**right_side**	**target**	

MS Windows only (Tk 8.3+):

no (no cursor)	**starting**	**size**	**size_ne_sw**	**size_ns**	**size_nw_se**	**size_we**	**uparrow**	**wait**

Mac only:

cross-hair	**ibeam**	**text**

3.20 Panedwindow

Command	Description
panedwindow *pathName ?options?*	(Tk 8.4+) Creates a panedwindow widget *pathName* with *options* and returns the new widget's path name. When invoked, *pathName* must not exist, but *pathName*'s parent should. A panedwindow widget can contain any number of panes arranged horizontally or vertically. Each pane contains one widget, and each pair of panes is separated by a moveable (via mouse movements) sash and sash handle. Moving a sash causes the widgets on either side of the sash to be resized. When a pane is resized from outside (eg, it is packed to expand and fill, and the containing toplevel is resized), space is added to the final (rightmost or bottommost) pane in the window.

Panedwindow Options

Standard

See Common Options and Resources in Options and Resources for full details.

-background	**-cursor**	**-relief**
-borderwidth	**-orient**	

Panedwindow Specific

See Coordinates in Options and Resources for screen unit options.

Configure Option	Resource Name	Resource Class	Description
-handlepad *size*	**handlePad**	**HandlePad**	Specifies the distance in screen units from the top or left end of the sash (depending on the orientation of the widget) at which to draw the handle.
-handlesize *size*	**handleSize**	**HandleSize**	Specifies the size of the square sash handle in screen units.
-height *height*	**height**	**Height**	Height of panedwindow in screen units. Default is to auto size.
-opaqueresize *boolean*	**opaqueResize**	**OpaqueResize**	Set to true, panes should be resized as a sash is moved, or if false, resizing should be deferred until the sash is placed.
-sashcursor *cursor*	**sashCursor**	**SashCursor**	Mouse cursor to use when over a sash. Default or when set to null, uses **sb_h_double_arrow** for horizontal and **sb_v_double_arrow** for vertical panedwindows.
-sashpad *size*	**sashPad**	**SashPad**	Specifies the amount of pad in screen units to leave on each side of a sash.
-sashrelief *relief*	**sashRelief**	**SashRelief**	Desired sash 3-D effect. Options are: **flat**, **groove**, **raised**, **ridge**, **solid** , or **sunken**.
-sashwidth *size*	**sashWidth**	**SashWidth**	Width of sash in screen units.
-showhandle *boolean*	**showHandle**	**ShowHandle**	Specifies whether sash handles should be shown.
-width *width*	**width**	**Width**	Width of panedwindow in screen units.

Panedwindow Commands

See Coordinates in Options and Resources for screen unit options.

Command	Description
pathName **add** *window ?window ...? ?option value ...?*	Add one or more *window*s to the panedwindow, each in a separate pane using the specified *options* . See **paneconfigure**for *options*. Tk 8.4.0 to 8.4.3 will set last pane to use all avalable space.
pathName **cget** *option*	Returns the current value of the configuration *option*. See Panedwindow Options above for *options* .
pathName **configure** *?option? ?value? ?option value ...?*	Change the configuration*option* to*value*. Without *value*, a list describing *option* is returned. Without *option*, a list of all available options for *pathName* is returned. For multiple options an empty string is returned. See Panedwindow Options above for *options*.
pathName **forget** *window ?window ...?*	Removes and unmaps each pane *window* from the panedwindow and forgets their configuration options.
pathName **identify** *x y*	Identify the panedwindow component at window coordinate *xy*. If over a slash or handle, returns two element list with index of slash or handle and type (slash or handle), else returns empty list. and
pathName **proxy** *?args?*	Used to query and change the position of the sash proxy for rubberband-style pane resizing. Valid *args* are:
coord	Return a list containing the x and y coordinates of the most recent proxy location.
forget	Remove the proxy from the display.
place *x y*	Place the proxy at the given *x* and *y* coordinates.
pathName **sash** *?args?*	Used to query and change the position of sashes in the panedwindow. Valid *args* are:

coord *index*	Return the current *x* and *y* coordinate pair for the top left corner of the region containing the sash given by *index*. *Index* must be an integer between 0 and 1 less than the number of panes in the panedwindow.
dragto *index x y*	Compute the difference between the given coordinates and the coordinates given to the last **sash coord** command for sash given by *index*. It then moves that sash the computed difference.
mark *index x y*	Records coordinates *x* and*y* for the sash given by *index*.
place *index x y*	Place the sash given by *index* at the coordinates *x* and *y*.
pathName **panecget** *window option*	Returns the current value of the configuration *option* for panewindow *window*. See **paneconfigure** below for *options*.
*pathName***paneconfigure** *window option ?value? ?option value ...?*	Change the configuration *option* to *value* for panewindow *window*. Without *value*, a list describing *option* is returned. Without *option*, a list of all available options for *pathName* is returned. For multiple options an empty string is returned.
-after *afterWindow*	Insert *window* managed by *pathName* after *afterWindow*.
-before *beforeWindow*	Insert *window* managed by *pathName* before *beforeWindow*.
-height *height*	Specify height of *window* in screen units (autosize is the default or when set to an empty string).
-hide boolean	(Tk 8.5+) Controls the visibility of a pane. Hidden panes are still maintained in the list of panes
-minsize *value*	Specifies the minimum size of window for the paned direction (vertical or horizontal) in screen units.
-padx *amount*	Specifies *amount* of horizontal padding to leave on each side of *window* in screen units (default is 0).
-pady *amount*	Specifies *amount* of vertical padding to leave on each side of *window* in screen units (default is 0).
-sticky *style*	Specifies where to position *window* in panewindow if the cavity is larger than the requested dimensions. *Style* can be zero or more positions (**n**, **s**, **e** or **w)** with optional space and comma separators. If both **n** and **s** (or **e** and **w**) are specified, the slave will be stretched to fill the entire height (or width) of its cavity. The default or when set to an empty string, is to center the slave within the cell.
-stretch when	(Tk 8.5+) Controls how extra space is allocated to each of the panes. Options are always (pane will always stretch), first (only left-most or top-most will stretch), last (only right-most or bottom-most will stretch), middle (will stretch if not the first or last), and never (pane will never stretch).
-width *width*	Specify width of *window* in screen units (autosize is the default or when set to an empty string).
*pathName***panes**	Returns an ordered list of the widgets managed by *pathName*.

3.21 Radiobutton

Command	Description
radiobutton *pathName ?options?*	Creates a radiobutton widget *pathName* with *options* and returns the new widget's path name. When invoked, *pathName* must not exist, but *pathName*'s parent should. A radiobutton widget displays a textual string, bitmap, or image and a diamond or circle called an *indicator*. By default a radiobutton is configured to select itself on a button click. To deselect a radiobutton, another button in the group must be selected. This means only one radiobutton within a group (all use same **-variable** variable) can be selected at a time. Radiobuttons also select and deselect themselves when the value of the **-variable** variable changes. Multiple fonts within a button text field are not supported.

Radiobutton Options

Standard

See Common Options and Resources in Options and Resources for full details.

-activebackground	**-disabledforeground**	**-padx**
-activeforeground	**-font**	**-pady**
-anchor	**-foreground**	**-relief**
-background	**-highlightbackground**	**-takefocus**
-bitmap	**-highlightcolor**	**-text**
-borderwidth	**-highlightthickness**	**-textvariable**
-compound (8.4+)	**-image**	**-underline**
-cursor	**-justify**	**-wraplength**

Radiobutton Specific

See Coordinates in Options and Resources for screen unit options.

Configure Option	Resource Name	Resource Class	Description
-command *script*	**command**	**Command**	Tcl command to associate with the button. *Script* is invoked when mouse button 1 is released over the button window. The button's global variable (**-variable** option) will be updated before the command is invoked.
-height *height*	**height**	**Height**	Height of button in screen units for bitmaps/images and in lines for text. Default is to auto size.
-indicatoron *boolean*	**indicatorOn**	**IndicatorOn**	Specifies whether the indicator should be drawn (default) or not. If false, the **-relief** option is ignored and the relief is set to sunken when widget is selected and raised in all other cases.
-offrelief *type*	**offRelief**	**OffRelief**	(Tk 8.4+) Specifies the relief for the radiobutton when the indicator is not drawn and the radiobutton is off. Options are: **flat**, **raised** (default), and **sunken**.
-overrelief *type*	**overRelief**	**OverRelief**	(Tk 8.4+) Alternative relief for when mouse cursor is over button. Not used when set to empty string (default). Options are: **flat**, **raised**, and **sunken**.
-selectcolor *color*	**selectColor**	**Background**	Specifies a background color to use when the button is selected. If set to empty string, no special color is used. If **-indicatoron** is true then the color applies to the indicator, if false this color is used as the background for the entire widget when selected.
-selectimage *image*	**selectImage**	**SelectImage**	Specifies image to be displayed when radiobutton is selected. Used with **-image**.
-state *state*	**state**	**State**	State of button. Options are: **active** (mouse pointer over button, use **activeforeground** and **activebackground**),**disabled** (button is insensitive, use **disabledforeground** and **background**), or **normal** (use **foreground** and **background**).
-tristateimage image	tristateImage	TristateImage	(Tk 8.5+) Specifies an image to display (in place of the image option) when the radiobutton is in tri-state mode. This option is ignored unless the image option has been specified.
-tristatevalue value	tristateValue	Value	(Tk 8.5+) Specifies the value that causes the radiobutton to display the multi-value selection, also known as the tri-state mode. Defaults to {}.
-value *value*	**value**	**Value**	Value stored in variable specified with **-variable** option when the radiobutton is selected.
-variable *variable*	**variable**	**Variable**	Specifies name of global variable to use for button selection status. Default is variable **selectedButton** .
-width *width*	**width**	**Width**	Width of button in screen units for bitmaps/images and in characters for text. Default is to auto size.

Effect	Options
Toolbar buttons	**-relief flat -overrelief raised**
Text-align toolbar buttons	**-offrelief flat -indicatoron false -overrelief raised**

Radiobutton Commands

Command	Description
pathName **cget** *option*	Returns the current value of the configuration *option* for radiobutton *pathName*. See Radiobutton Widget Options above for *options*.
pathName **configure***?option? ?value? ?option value ...?*	Change the configuration *option* for the radiobutton *pathNamevalue*. Without *value*, a list describing the available options is returned. Without *option*, a list describing all of the available options for *pathName* is returned. For multiple options an empty string is returned. See Radiobutton Widget Options to above for *options*.
pathName **deselect**	Deselect the radiobutton and set the associated variable to its "off" value of empty string.
pathName **flash**	Flash radiobutton by toggling between active and normal colors several times. Radiobutton is left is initial state of **active** or **normal**. Ignored if radiobutton is disabled.
pathName **invoke**	Selects the radiobutton and invokes the Tcl command specified with **-command**, if any. Returns value of Tcl command or empty string if no **-command**. Ignored if button is disabled.
pathName **select**	Selects the radiobutton and set the associated variable to its "on" value.

Default Radiobutton Bindings

Active or normal radiobutton default bindings:

Event	Description
<Enter>	On Unix, when mouse passes over button **state**becomes**active**.
<Leave>	On Unix, when mouse leaves the button **state** becomes **normal**.
<Button-1> or **<Return>****<space>** or	On Unix, **relief** changes to **sunken** and associated **-command** *script* is executed.
<Button-1>	On Windows and Mac, **relief** changes to **sunken** and state becomes **active**.
<ButtonRelease-1>	On Windows and Mac, **relief** changes to **raised**, state becomes **normal**, and associated **-command** *script* is executed.
<Enter>	On Windows and Mac, **relief** changes to **sunken** and state becomes **active**.
<space>	On Windows and Mac, **relief** changes to **sunken** and associated **-command** *script* is executed.

3.22 Scale Widget

Command	Description
scale *pathName ?options?*	Creates a scale widget *pathName* with *options* and returns the new widget's path name. When invoked, *pathName* must not exist, but *pathName*'s parent should.A scale widget displays a rectangular trough and a small slider either a vertical or horizontal orientation. The scale value may be linked to the slider, such that a change in one affects the other.

Scale Widget Options

Standard

See Common Options and Resources in Options and Resources for full details.

-activebackground	**-foreground**	**-relief**
-background	**-highlightbackground**	**-repeatdelay**
-borderwidth	**-highlightcolor**	**-repeatinterval**
-cursor	**-highlightthickness**	**-takefocus**
-font	**-orient**	**-troughcolor**

Scale Widget Specific

See Coordinates in Options and Resources for screen unit options.

Configure Option	Resource Name	Resource Class	Description
-bigincrement *number*	**bigIncrement**	**BigIncrement**	Specifies the increment size for interactions with scale that cause its value to change by "large" increments. A value of 0 sets the large increments default to 1/10 the range of the scale.
-command *tclCommand*	**command**	**Command**	Tcl command to execute when scale's value changes via widget command. Passes new scale value as an arg.
-digits *integer*	**digits**	**Digits**	An integer specifying how many significant digits should be retained when converting the value of the scale to a string. If <= 0, scale picks the smallest value forwhich each slider position prints a different string.
-from *number*	**from**	**From**	A real value specifying the left or top end of the scale.
-label *string*	**label**	**Label**	A string to display as the label at the top right of the scale for vertical scales and at the top left of the scale for horizontal scales.
-length *size*	**length**	**Length**	Specifies the desired long dimension (height for vertical or width for horizontal) of the scale in screen units.
-resolution *number*	**resolution**	**Resolution**	A real value (default is 1) specifying the scale resolution. When *number* > 0, the scale's value, tick marks, and endpoints will be rounded to an even multiple of *number* . When *number* < 0, no rounding occurs.
-showvalue *boolean*	**showValue**	**ShowValue**	Specifies whether to show the value of the scale to the left of the slider for vertical scales or above the slider for horizontal scales.
-sliderlength *size*	**sliderLength**	**SliderLength**	Specifies long dimension size of the slider in screen units.
-sliderrelief *relief*	**sliderRelief**	**SliderRelief**	Specifies the relief to use for the slider. Options are: **flat**, **groove**, **raised**, **ridge** ,**solid**, or **sunken**.
-state *state*	**state**	**State**	State of button. Options are: **active** (use **activebackground**),**disabled** (value can't be changed), or **normal** (use **background**).
-tickinterval *number*	**tickInterval**	**TickInterval**	A real value specifying the spacing between tick marks placed to the left of the trough for vertical scales and below the trough for horizontal scales. Set to 0 for no tick marks.
-to *number*	**to**	**To**	A real value specifying the right or bottom end of the scale.
-variable *variable*	**variable**	**Variable**	Specifies name of global variable to use for scale value.
-width *width*	**width**	**Width**	Specifies the desired narrow dimension (width for vertical or height for horizontal) of the scale in screen units.

Scale Elements

Element	Description
trough1	Region between the slider and top or left end of scale.
slider	Rectangle that indicates value or position of scale.
trough2	Region between the slider and bottom or right end of scale.

Scale Commands

Command	Description
pathName **cget** *option*	Returns the current value of the configuration *option* for scale *pathName*. See Scale Widget Options above for *options*.
pathName **configure***?option? ?value? ?option value ...?*	Change the configuration *option* for the scale *pathName* to *value*. Without *value*, a list describing the available options is returned. Without *option*, a list describing all of the available options for *pathName* is returned. For multiple options an empty string is returned. See Scale Widget Options above for *options*.
pathName **coords** *?value?*	Returns a list of the *x* and *y* coordinates of the point along the centerline of the scale corresponding to *value* (default is scale's current value).
pathName **get** *?x y?*	Returns the scale value corresponding the coordinate *x* and *y*. Default is to return the scale's current value.
pathName **identify** *x y*	Returns a string indicating what part of scale is at coordinate *x* and *y*. Valid values are **empty** (not a valid element) or one of the Scale Elements above.
pathName **set** *value*	Changes the current value of scale to *value* .

Scale Bindings

Event	Description
<Enter>	Activate scale.
<Motion>	Activate scale.
<Leave>	Deactivate scale.
<Button-1>	If in trough, scale's value will be incremented or decremented by value of **-resolution** option in the direction of the button press. If the button is held down, the action auto-repeats.
<ButtonRelease-1>	Cancel repeat, end drag, and activate scale.
<B1-Leave> or **<B1-Enter>**	No function
<B1-Motion>	If pressed over the slider, the slider can be dragged with the mouse.
<Control-Button-1>	If in trough, slider moves all the way to the end of its range in the direction of the button press.
<Button-2>	Scale's value is set to the mouse position.
<B2-Motion>	Scale's value changes with the drag.
<ButtonRelease-2>	Cancel repeat, end drag, and activate scale.
<B2-Leave> or **<B2-Enter>**	No function
<Up> or **<Left>**	Move the slider up or left by the value of the **-resolution** option.
<Down> or **<Right>**	Move the slider down or right by the value of the **-resolution** option.
<Control-Up> or **<Control-Left>**	Move the slider up or left by the value of the **-bigincrement** option.
<Control-Down> or **<Control-Right>**	Move the slider down or right by the value of the **-bigincrement** option.
<Home>	Moves the slider to the top or left end of its range.
<End>	Moves the slider to the bottom or right end of its range.

3.23 Scrollbar

Command	Description
scrollbar *pathName ?options?*	Creates a scrollbar widget *pathName* with *options* and returns the new widget's path name. When invoked, *pathName* must not exist, but *pathName*'s parent should. A scrollbar widget displays two arrows, one at each end of the scrollbar, and a *slider* in the middle portion of the scrollbar. It provides a visual representation of how much of an *associated window* is visible and also a way to change the visible portion.

Scrollbar Widget Options

Standard

See Common Options and Resources in Options and Resources for full details.

-activebackground	**-highlightcolor**	**-repeatdelay**
-background	**-highlightthickness**	**-repeatinterval**
-borderwidth	**-jump**	**-takefocus**
-cursor	**-orient**	**-troughcolor**
-highlightbackground	**-relief**	

Scrollbar Widget Specific

See Coordinates in Options and Resources for screen unit options.

Configure Option	Resource Name	Resource Class	Description
-activerelief *number*	**activeRelief**	**ActiveRelief**	Relief to use for active element. Options are: **flat**, **groove**, **raised**,**ridge**, **solid**, or **sunken**. Non-active elements use the **raised** relief.
-command *tclCommand*	**command**	**Command**	Tcl command to invoke to change the view in the widget associated with the scrollbar. See Scrolling Commands below for args passed to *tclCommand*.
-elementborderwidth *width*	**elementBorderWidth**	**BorderWidth**	Specifies width of borders around internal elements (arrows and slider) in screen units. If set to 0, **-borderwidth** is used instead.
-width *width*	**width**	**Width**	Specifies the desired narrow dimension (width for vertical or height for horizontal) of the scrollbar in screen units.

Scrollbar Elements

Element	Description
arrow1	Top or left arrow in the scrollbar.
trough1	Region between the slider and **arrow1**.
slider	Rectangle that indicates what is visible in the associated widget.
trough2	Region between the slider and **arrow2**.
arrow2	Bottom or right arrow in the scrollbar.

Scrollbar Commands

Command	Description
pathName **activate** *?element?*	Marks *element* as the active element. Except for the troughs, *element* can be one of the elements listed in Scrollbar Elements above. Without *element*, returns the current active element or empty string if none.
pathName **cget** *option*	Returns the current value of the configuration *option* for scrollbar *pathName*. See Scrollbar Widget Options above for *options*.
pathName **configure***?option? ?value? ?option value ...?*	Change the configuration *option* for the scrollbar *pathName* to *value*. Without *value*, a list describing the available options is returned. Without *option* , a list describing all of the available options for *pathName* is returned. For multiple options an empty string is returned. See Scrollbar Widget Options above for *options*.
pathName **delta** *deltaX deltaY*	Returns a real number (-1 to 1) indicating the change in the scrollbar setting corresponding to the *deltaX* (horizontal scrollbar) or *deltaY* (vertical scrollbar) value in pixels. The args and the result may be zero or negative.
pathName **fraction** *x y*	Returns a real number (0 to 1) indicating where the closest point given by pixel coordinate *x* and *y* lies in the trough area of the scrollbar. Top or left is at 0 and the bottom or right is at 1.
pathName **get**	Returns current scrollbar settings as the list whose elements are the args to the most recent **set** widget command.
pathName **identify** *x y*	Returns the name of element under pixel coordinates *x* and *y* or empty string if none. See Scrollbar Elements above valid elements.
pathName **set** *first last*	Invoked by scrollbar's associated widget to describe the current view in the widget. *First* and *last* are real values (0 to 1) describing the first and last part of the visible portion of the scrollbar's associated widget.

Scrollbar Commands

The following are the valid formats of the command invoked by the **-command** option to notify the scrollbar's associated widget to change its view. The *pathName* is the scrollbar's associated widget and *command* is either **xview** (for horizontal scrollbars) or **yview** (for vertical scrollbars).

Command	Description
pathName command **moveto** *fraction*	Widget should adjust its view so that the point given by real number *fraction* (0 to 1) appears at the beginning of the widget.
pathName command **scroll** *number* **units**	Widget should adjust its view by*number* units (characters or lines for text widgets or screen units for bitmaps or images).
pathName command **scroll** *number* **pages**	Widget should adjust its view by*number* pages (height of the window or screenful, etc.).

Scrollbar Bindings

Event	Over Element	Description
<Enter>		Activate scrollbar.
<Motion>		Activate scrollbar.
<Leave>		Deactivate scrollbar.
<Button-1>	arrow1	Shifts view in the associated widget up or to the left by one unit so document appears to move down or to the right. If the button is held down, the action auto-repeats.
<Button-1>	trough1	Shifts view in the associated widget up or to the left by one screenful so document appears to move down or to the right. If the button is held down, the action auto-repeats.
<Button-1>	trough2	Shifts view in the associated widget down or to the right by one screenful so document appears to move up or to the left. If the button is held down, the action auto-repeats.
<Button-1>	arrow2	Shifts view in the associated widget down or to the right by one unit so document appears to move up or to the left. If the button is held down, the action auto-repeats.
<B1-Motion>	slider	View changes as the slider is dragged. If the **jump** option is true, the view only changes when the mouse button is released.
<Button-2>	trough or slider	Sets the view to correspond to the mouse position.
<Button-2>	arrow	Same as **<Button-1>**.
<B2-Motion>	trough or slider	Causes the view to drag with the mouse.
<Control-Button-1>	arrow1 or trough1	Adjusts view to the very top or left of the document.
<Control-Button-1>	arrow2 or trough2	Adjusts view to the very bottom or right of the document.
<Up>	any	For vertical scrollbars, shifts view in the associated widget up by one unit so document appears to move down. If the key is held down, the action auto-repeats.
<Down>	any	For vertical scrollbars, shifts view in the associated widget down by one unit so document appears to move up. If the key is held down, the action auto-repeats.
<Left>	any	For horizontal scrollbars, shifts view in the associated widget left by one unit so document appears to move right. If the key is held down, the action auto-repeats.
<Right>	any	For horizontal scrollbars, shifts view in the associated widget right by one unit so document appears to move left. If the key is held down, the action auto-repeats.
<Control-Up>	any	For vertical scrollbars, shifts view in the associated widget up by one screenful so document appears to move down. If the keys are held down, the action auto-repeats.
<Control-Down>	any	For vertical scrollbars, shifts view in the associated widget down by one screenful so document appears to move up. If the keys are held down, the action auto-repeats.
<Control-Left>	any	For horizontal scrollbars, shifts view in the associated widget left by one screenful so document appears to move to the right. If the keys are held down, the action auto-repeats.
<Control-Right>	any	For horizontal scrollbars, shifts view in the associated widget to the right by one screenful so document appears to move to the left. If the keys are held down, the action auto-repeats.
<Prior>		Shifts view in the associated widget up or to the left by one screenful so document appears to move down or to the right. If the button is held down, the action auto-repeats.
<Next>		Shifts view in the associated widget down or to the right by one screenful so document appears to move up or to the left. If the button is held down, the action auto-repeats.
<Home>		Adjusts view to the very top or left of the document.
<End>		Adjusts view to the very bottom or right of the document.

3.24 Spinbox Widget

Command	Description
spinbox *pathName ?options?*	(Tk 8.4+) Creates a spinbox widget *pathName* with *options* and returns the new widget's path name. When invoked, *pathName* must not exist, but *pathName*'s parent should. A spinbox consists of an editable entry field and two arrow button to move, or spin, through a fixed set of ascending or descending values.

Spinbox Widget Options

Standard

See Common Options and Resources in Options and Resources for full details.

-activebackground	**-highlightthickness**	**-repeatinterval**
-background	**-insertbackground**	**-selectbackground**
-borderwidth	**-insertborderwidth**	**-selectborderwidth**
-cursor	**-insertofftime**	**-selectforeground**
-exportselection	**-insertontime**	**-takefocus**
-font	**-insertwidth**	**-textvariable**
-foreground	**-justify**	**-xscrollcommand**
-highlightbackground	**-relief**	
-highlightcolor	**-repeatdelay**	

Spinbox Widget Specific

See Coordinates in Options and Resources for screen unit options. See Colors in Options and Resources for color formats.

Configure Option	Resource Name	Resource Class	Description
-buttonbackground *color*	**buttonBackground**	**Background**	Specifies background color for the spin buttons.
-buttoncursor *cursor*	**buttonCursor**	**Cursor**	Specifies cursor to use when over the spin buttons. If set to an empty string (default), the default cursor will be used. See Cursors in Options and Resources for options.
-buttondownrelief *relief*	**buttonDownRelief**	**Relief**	Specifies relief for the upper spin button. Options are: **flat**, **groove**, **raised**,**ridge**, **solid**, or **sunken**.
-buttonuprelief *relief*	**buttonUpRelief**	**Relief**	Specifies relief for the lower spin button. Options are: **flat**, **groove**, **raised**,**ridge**, **solid**, or **sunken**.
-command *script*	**command**	**Command**	Tcl command to invoke when spinbutton is invoked. Recognizes %W, %s, and %d substitutions.
-disabledbackground *color*	**disabledBackground**	**DisabledBackground**	Background color of widget when the spinbox is disabled. If set to the empty string, the normal background color is used.
-disabledforeground *color*	**disabledForeground**	**DisabledForeground**	Foreground color of widget when the spinbox is disabled. If set to the empty string, the normal foreground color is used.
-format *format*	**format**	**Format**	Specifies alternate format for setting string value. *Format* is %#.#f.
-from *value*	**from**	**From**	Specifies lowest floating point value for spinbox.

-invalidcommand *script*	**invalidCommand**	**InvalidCommand**	Specifies script to eval when **-validcommand** returns 0. If set to the empty string (default), disables option. Typically set to **bell**. See Percent Substitutions below for valid % substitutions. (Also **-invcmd**).
-increment *value*	**increment**	**Increment**	Specifies floating poiunt value to add to or subtract from the spinbox's value when the buttons are selected.
-readonlybackground *color*	**readonlyBackground**	**ReadonlyBackground**	Background color of widget when the spinbox is read-only. If set to the empty string, the normal background color is used.
-state *state*	**state**	**State**	State of button. Options are: **disabled** (cannot change or select contents, use **disabledforeground** and **disabledbackground**),**normal** (can change and select contents, use **foreground** and **background**),**readonly** (cannot change but can select contents, use **foreground** and **readonlybackground**).
-to *value*	**to**	**To**	Specifies highest floating point value for spinbox.
-validate *mode*	**validate**	**Validate**	Specifies validation mode. See Validation Types below for options.
-validatecommand *script*	**validateCommand**	**ValidateCommand**	Specifies script to eval when spinbox input is to be validated. If set to the empty string (default), disables option. Script must return 1 to accept or 0 to reject new value. See Percent Substitutions below for valid % substitutions. (Also **-vcmd**).
-values *valueList*	**values**	**Values**	Specifies list of valid values for spinbox.
-width *width*	**width**	**Width**	Width of spinbox window in font average-sized characters. If <=0, auto size based on current text.
-wrap *boolean*	**wrap**	**Wrap**	Specifies whether values larger than spinbox are wrapped.

Validation Types

Type	Description
none	Do not perform validation (default).
focus	**-validatecommand** will be called when the spinbox receives or loses focus.
focusin	**-validatecommand** will be called when the spinbox receives focus.
focusout	**-validatecommand** will be called when the spinbox loses focus.
key	**-validatecommand** will be called when the spinbox is edited.
all	**-validatecommand** will be called for all above conditions.

Percent Substitutions

Sub	Description
%d	Type of action: 1 for insert, 0 for delete, or -1 for focus, forced, or textvariable validation.
%i	Index of char string to be inserted/deleted, if not -1.
%P	The value of the spinbox should **-validatecommand** accept the new value. When configuring to a new textvariable, this will be the value of that textvariable.
%s	The current value of the spinbox before **-validatecommand** accepts the new value.
%S	The text string being inserted/deleted, if not an empty string {}.
%v	The current validation type (**none**, **focus** ,**focusin**,**focusout**,**key**, or **all**).
%V	The type of validation that triggered the callback (**key**,**focusin**,**focusout**,**forced**).
%W	The name of the spinbox widget.

Indicies or Character Positions

Some spinbox commands support the use of an index to locate the position of characters within the spinbox string starting from 0. The following are the valid forms of specifying an *index*:

Index form	Description
number	A decimal number giving the position or index (starting from 0) of the desired character within the spinbox string. If *number* < 0, the 0 is used, if *number* > length of text list, then **end** is used.
anchor	Selection anchor point as set by the **select from** and **select adjust** commands.
end	Character or coordinate just after last one in spinbox's string.
insert	Character just after the insertion cursor.
sel.first	First character in selection. Returns an error if selection is not in the spinbox.
sel.last	Character just after last character in selection. Returns an error if selection is not in the spinbox.
@*number*	Character at the x-coordinate point in the spinbox's window. If *x* is outside the spinbox window's range, it is set to the nearest legal value.

Spinbox Widget Commands

Command	Description
pathName **bbox***index*	Returns a list of four elements *x y w h* , giving an approximate bounding box for the character at position *index* . Coordinates *x*,*y* are top-left corner of character at *index*,*w* is width of char, and *h* is height of char in pixels.
pathName **cget** *option*	Returns the current value of the configuration *option*. See Spinbox Widget Options above for *options* .
pathName **configure** *?option? ?value? ?option value ...?*	Change the configuration *option* to*value*. Without *value*, a list describing *option* is returned. Without *option*, a list of all available options for *pathName* is returned. For multiple options an empty string is returned. See Spinbox Widget Options above for *options*.
pathName **delete** *first ?last?*	Delete characters in spinbox's string from position *first* up to but not including position *last* (default is *first*+1 to delete 1 character). See Indicies or Character Positions above for *first* and *last* options.
pathName **get**	Returns the spinbox's string.
pathName **icursor** *index*	Display the insertion cursor just before the character at position *index*. See Indicies or Char Positions above for index options.
pathName **identify** *x y*	Returns the name of the window element at position *x* and *y* in the spinbox. Options are: **none** ,**buttondown**,**buttonup**, or **entry**.
pathName **index** *index*	Returns the numerical index corresponding to *index*. See Indicies or Character Positions above for index options.

pathName **insert** *index string*	Insert *string* just before the character at position *index*. See Indicies or Character Positions above for index options.
pathName **invoke** *element*	Invokes the specified element, where *element* is **buttondown** or **buttonup**.
pathName **scan** *option args*	Implements scanning on spinbox widgets. *Options* are:
mark *x*	Records *x* and the current view in the spinbox window. Typically associated with mouse button press in widget.
dragto *x*	Adjusts the view by 10 times the difference between the coordinate *x***mark** *x* coordinate. Used with mouse motion events to produce high speed dragging. and the last
pathName **selection** *option arg*	Manipulates the selection within an spinbox based on *option*. See Indicies or Char Positions above for *index* options. Vaild *options* and *args* are:
adjust *index*	Adjust the end of the selection nearest to the character given by position *index* to include characters up to *index* and set the other end to be the anchor point. Works the same as **selection to** if selection is not in spinbox widget.
clear	Clear the selection if it is in the widget.
element *?element?*	Sets the current selection to *element*. Without *element*, returns the currently selected element.
from *index*	Sets the selection anchor point to the character given by position *index*.
present	Returns 1 if characters are selected in the spinbox, 0 if not.
range *start end*	Sets the selection to include characters from position *start* up to but not including position *end*.
to *index*	If *index* < anchor point, set the selection to include characters from position *index* up to but not including the anchor point. If *index* > anchor point, set the selection to include characters from the anchor point up to but not including position *index*. If *index* = anchor point, no change is made. If the selection isn't in the spinbox widget, use the most recent anchor point specified for the widget.
pathName **set** *?string?*	Sets spinbox to *string*. Without *string*, returns the current spinbox's string.
*pathName***validate**	Forces the evaluation of **-validatecommand** by temporarily setting **validate** to **all** and returns result.
pathName **xview** ?*option args*?	Query or change the horizontal view of the spinbox. Without any *options*, returns a two element list specifying the start and end of the visible fraction (from 0 to 1) of the horizontal span of the widget between the left and right edges of the window. Vaild *options* and *args* are:
index	Adjust window view to display the character at position *index* at the left edge of window. See Indicies or Char Positions above for *index* options.
moveto *fraction*	Adjust window view so that *fraction* (from 0 to 1) of the total width of the widget is off-screen to the left.
scroll *number* **pages**	Shift the view left (*number* < 0) or right (*number* > 0) by *number* screenfuls.
scroll *number* **units**	Shift the view left (*number* < 0) or right (*number* > 0) by *number* average-width characters.

Default Spinbox Widget Bindings

For additional default bindings see Virtual Events in Bindings and Virtual Events.

Event	Description
<Button-1>	Positions the insertion cursor just before the character underneath the mouse cursor, sets the input focus to this widget, and clears any selection in the widget.
<B1-Motion>	Drags out a selection (in words if double clicked) between the insertion cursor and the character under the mouse.
<Double-Button-1>	Selects the word under the mouse and positions the insertion cursor at the beginning of the word.
<Triple-Button-1>	Selects all of the text in the spinbox and positions the insertion cursor before the first character.
<Shift-B1-Motion>	Adjusts the end of the selection (in words if double clicked) that was nearest to the mouse cursor when button 1 was pressed.
<Control-Button-1>	Position the insertion cursor in the spinbox without affecting the selection.
<B1-Leave>	Adjusts view in spinbox left or right more quickly.
<B1-Enter>	Stops adjustment of view in spinbox left or right more quickly.
<Button-2>	Paste selection into the spinbox at the position of the mouse cursor.
<B2-Motion>	Adjusts view in spinbox by scrolling left or right.
<Left> or **<Control-b>**	Moves the insertion cursor one character back (left), clears any selection in the spinbox, and sets the selection anchor.
<Right> or **<Control-f>**	Moves the insertion cursor one character forward (right), clears any selection in the spinbox, and sets the selection anchor.
<Shift-Left>	Move the insertion cursor one character back (left) and extend the selection to include the new character.
<Shift-Right>	Move the insertion cursor one character forward (right) and extend the selection to include the new character.
<Control-Left> or **<Meta-b>**	Move the insertion cursor back (left) by one word, clears any selection in the spinbox, and sets the selection anchor.
<Control-Right> or **<Meta-f>**	Move the insertion cursor forward (right) by one word, clears any selection in the spinbox, and sets the selection anchor.
<Shift-Control-Left>	Move the insertion cursor back (left) by one word and also extend the selection.
<Shift-Control-Right>	Move the insertion cursor forward (right) by one word and also extend the selection.
<Home> or **<Control-a>**	Move the insertion cursor to the beginning of the spinbox and clear any selection in the spinbox.
<Shift-Home>	Move the insertion cursor to the beginning of the spinbox and also extends the selection to that point.
<End> or **<Control-e>**	Move the insertion cursor to the end of the spinbox and clear any selection in the spinbox.
<Shift-End>	Move the insertion cursor to the end of the spinbox and also extends the selection to that point.
<Select> or **<Control-Space>**	Set the selection anchor to the position of the insertion cursor without affecting the selection.
<Shift-Select> or **<Shift-Control-Space>**	Adjusts the selection to the current position of the insertion cursor, selecting from the anchor to the insertion cursor if there is not an existing selection.
<Control-slash>	Selects all the text in the spinbox.
<Control-backslash>	Clears any selection in the spinbox.
<Delete>	Deletes the selection, if there is one in the spinbox, if not it deletes the character to the right of the insertion cursor.
<BackSpace> or **<Control-h>**	Deletes the selection, if there is one in the spinbox, if not it deletes the character to the left of the insertion cursor.
<Control-d>	Deletes the character to the right of the insertion cursor.
<Meta-d>	Deletes the word to the right of the insertion cursor.
<Control-k>	Deletes all the characters to the right of the insertion cursor.
<Control-t>	Reverses the order of the two characters to the right of the insertion cursor.
<Keypress>	Insert character into spinbox widget.

3.25 Text Widget

Command	Description
text *pathName ?options?*	Creates a text widget *pathName* with *options* and returns the new widget's path name. When invoked, *pathName* must not exist, but *pathName*'s parent should. A text widget displays one or more lines of text and can allow that text to be edited.
tk_textCopy *pathName*	(Tk 8.4+) Copies the selection in text widget *pathName* to the clipboard.
tk_textCut *pathName*	(Tk 8.4+) Copies the selection in text widget *pathName* to the clipboard and deletes it from the text widget.
tk_textPaste *pathName*	(Tk 8.4+) Inserts the contents of the clipboard into text widget *pathName* at the position of the insertion cursor.

Text Widget Options

Standard

See Common Options and Resources in Options and Resources for full details.

-background	**-highlightthickness**	**-relief**
-borderwidth	**-insertbackground**	**-selectbackground**
-cursor	**-insertborderwidth**	**-selectborderwidth**
-exportselection	**-insertofftime**	**-selectforeground**
-font	**-insertontime**	**-setgrid**
-foreground	**-insertwidth**	**-takefocus**
-highlightbackground	**-padx**	**-xscrollcommand**
-highlightcolor	**-pady**	**-yscrollcommand**

Text Widget Specific

See Coordinates in Options and Resources for screen unit options.

Configure Option	Resource Name	Resource Class	Description
-autoseparators *boolean*	**autoSeparators**	**AutoSeparators**	(Tk 8.4+) Specifies whether separators are automatically inserted in the undo stack. Used with **-undo**. Common to all peers.
-blockcursor	blockCursor	BlockCursor	(Tk 8.5+) Specifies whether the insertion cursor should be drawn as a block (true) or thin vertical line (false or default).
-endline	endLine	EndLine	(Tk 8.5+) Specifies index of the last line of the underlying textual data store that should be shown. Default is {}, which sets the end to after the last line.
-height *size*	**height**	**Height**	Height of text widget in lines of **-font** sized text.
-inactiveselectionbackground	inactiveSelectionBackground	Foreground	(Tk 8.5+) Specifies color of the selection, or {} for no selection, when the window does not have the input focus.
-maxundo *count*	**maxUndo**	**MaxUndo**	(Tk 8.4+) Specifies the max number of compound undo actions on the undo stack. If *count* <= 0, use an unlimited undo stack. Common to all peers.
-spacing1 *size*	**spacing1**	**Spacing1**	Space in screen units above first line of a paragraph.
-spacing2 *size*	**spacing2**	**Spacing2**	Space in screen units between lines within a paragraph.
-spacing3 *size*	**spacing3**	**Spacing3**	Space in screen units below the last line of a paragraph.
-startline	startLine	StartLine	(Tk 8.5+) Specifies index of the first line of the underlying textual data store that should be shown. Default is {}, which sets the start to before the first line.
-state *state*	**state**	**State**	State of text widget. Options are: **disabled** or **normal**.
-tabs *tabList*	**tabs**	**Tabs**	Specifies a list of tab stops consisting of offset values from the left edge in screen units followed by an optional justification of either **left** (default) with the left edge of text at tab position, **right** with text at tab position, **center** with the text centered at the tab position, or **numeric** with decimal point in the text is positioned at the tab position. If set to **{ }** the default 8-character tab stops are used.
-undo *boolean*	**undo**	**Undo**	(Tk 8.4+) Specifies whether the undo mechanism is active. Common to all peers.
-width *size*	**width**	**Width**	Width of text widget in **-font** sized characters. For proportional fonts, width of "0" is used.
-wrap *type*	**wrap**	**Wrap**	Specifies how to wrap lines wider than the window. Options are: **char** (line break can be made after any character), **none** (no wrap), or **word** (line break can only be made at word boundaries).

Indicies or Character Positions:

Some text widget commands support the use of an index to locate the position of characters within the text widget. Indicies have the syntax:
base modifier modifier modifier ...
The following are the valid forms for *base*:

Base	Description
line.char	Indicates line *line* (starts at 1) and character *char* (starts at 0).
@x,y	Indicates the character that covers the pixel at position *x* and *y*.
end	Indicates the character at the end of the text, just after the newline.
mark	Indicates the character just after the mark whose name is *mark*.
*tag***.first**	Indicates the first character in the text that has been tagged with *tag*. If no characters are tagged, an error will be generated.
*tag***.last**	Indicates the character just after the last one in the text that has been tagged with *tag*. If no characters are tagged, an error will be generated.
pathName	Indicates the position of the embedded window whose name is *pathName*. If *pathName* doesn't exist, an error is generated.
imageName	Indicates the position of the embedded image whose name is *imageName*. If *imageName* doesn't exist, an error is generated.

The following are the valid forms for *modifier*:

Modifier	Description
+count ?submodifer? **chars**	Adjust the index forward by *count* characters, moving to later lines in the text if necessary or to the last character in the text if fewer than *count* characters remain. In Tk 8.5+, use the display submodifier to skip and not count elided characters and any to count all characters (default).
-count ?submodifer? **chars**	Adjust the index backwards by *count* characters, moving to earlier lines in the text if necessary or to the first character in the text if fewer than *count* characters remain. In Tk 8.5+, use the submodifier display to skip and not count elided characters and any to count all characters (default).
+count ?submodifer? indicies	(Tk 8.5+) Adjust the index forward by count index positions, moving to later lines in the text if necessary. If there are fewer than count index positions in the text after the current index, then set the index to the last index position in the text. In Tk 8.5+, use the submodifier display to skip and not count elided indicies and any to count all indicies (default).
-count ?submodifer? indicies	(Tk 8.5+) Adjust the index backward by count index positions, moving to earlier lines in the text if necessary. If there are fewer than count index positions in the text before the current index, then set the index to the first index position (1.0) in the text. In Tk 8.5+, use the submodifier display to skip and not count elided indicies and any to count all indicies (default).
+count ?submodifer? **lines**	Adjust the index forward by *count* lines or to the last line if less than *count* remain, without changing the character position within the line or to the last character in the line (newline char) if fewer characters than the character position are available. In In Tk 8.5+, use the submodifier display to count visual lines and any to count logical lines (default).
-count ?submodifer? **lines**	Adjust the index backwards by *count* lines or to the first line if less than *count* remain, without changing the character position within the line or to the last character in the line (newline char) if fewer characters than the character position are available. In In Tk 8.5+, use the submodifier display to count visual lines and any to count logical lines (default).
linestart	Adjust the index to refer to the first character on the line.
lineend	Adjust the index to refer to the last character on the line (newline char).
wordstart	Adjust the index to refer to the first character of the word (consists of letters, digits, underscores, or any other single char) containing the current index.
wordend	Adjust the index to refer to the last character of the word (consists of letters, digits, underscores, or any other single char) containing the current index.

Annotations

Annotation	Description
Tag	Tags are a textual string identifiers that can be associated with a single character, range of characters, or several ranges of characters in the text widget. There can be an unlimited number of tags within a text widget and any number associated with any particular character. Deleting a character also removes the tag for that text. The default prority order for tags is based on the order defined, with the latest having the highest prioity. When tags conflict, the tag with the highest priority is used. See Tag Options below. The **sel** tag is associated with the current selection if the **-exportSelection** option is true. The **sel** tag can not be deleted. See Selection Support below. The sel tag may be set and configured (in its display style) differently for each peer.
Mark	Marks are textual strings that are used as floating markers in the text to keep track of particular places in the text as it is edited. Marks are associated with the gap between two characters and a single position can only be associated with one mark. Deleting the characters around a mark does not delete the mark. Marks have a gravity of **left** or **right** (default), which defines what happens to the mark (which text it stays with) when text is inserted at the point of the mark. The **insert** mark is associated with the insertion cursor and the**current** mark is associated with the character closest to the mouse pointer unless the mouse button is held down. The **insert** and **current** marks can not be deleted. Each peer has its own insert and current mark positions (but all other marks are shared)
Embedded Windows	Embedded windows allow any number of widgets to be embedded in a text widget which will dynamically update as the text is modified or scrolled. They will be mapped and unmapped when moved into and out of the visible area of the text widget. Each embedded window occupies one character's worth of index space in the text widget, and it may be referred to either by its name or by its position in the widget's index space. If the range of text containing the embedded window is deleted then the window is destroyed. See Embedded Window Options below. Embedded windows, which are arbitrary other widgets, cannot be shared between peers.
Embedded Images	Embedded images allow any number of images to be embedded in a text widget. An image may be embedded multiple times. The image positions will be updated as text is updated ot scrolled. Each embedded image occupies one character's worth of index space in the text widget, and it may be referred to either by its name or by its position in the widget's index space. If the range of text containing the embedded image is deleted then the image is removed. See Embedded Image Options below.

Tag Options

See Colors, Coordinates, or Default Bitmaps in Options and Resources for color, screen unit, and bitmap options, respectively. See Fonts for font options.

Option	Description
-background *color*	Specifies the background color to use for characters associated with the tag.
-bgstipple *bitmap*	Specifies a bitmap that is used as a stipple pattern for the background. A solid fill will be used as the default option or if set to an empty string.
-borderwidth *pixels*	Specifies the width of a 3-D border to draw around the background in screen units.
-elide *boolean*	(Tk 8.3+) Specifies whether the data should be elided. Elided data is not displayed and takes no space on screen, but further on behaves just as normal data.
-fgstipple *bitmap*	Specifies a bitmap that is used as a stipple pattern for the foreground. A solid fill will be used as the default option or if set to an empty string.
-font *fontName*	Specifies the name of the font to use for drawing characters.
-foreground *color*	Specifies the foreground color to use for characters associated with the tag.
-justify *justify*	Specifies how to justify text only if the first character in a line has a tag with this option. Options are **left**, **right**, or **center**.
-lmargin1 *size*	Specifies the left margin or indentation in screen units for the first line in a paragraph. The first character in the text line must have the tag in order to take effect.
-lmargin2 *size*	Specifies the left margin or indentation in screen units for the subsequent lines in a paragraph. The first character in the text line must have the tag in order to take effect.
-offset *size*	Specifies the amount in screen units by which the text's baseline should be offset vertically from the baseline of the overall line. Use a positive offset for superscripts and a negative offset for subscripts.
-overstrike *boolean*	Specifies whether to draw a horizontal rule through the middle of characters.
-relief *relief*	Specifies the 3-D relief to use for drawing backgrounds. Options are: **flat**, **groove**, **raised**, **ridge**, **solid**, or **sunken**.
-rmargin *size*	Specifies the right margin in screen units for lines in a paragraph. The first character in the text line must have the tag in order to take effect.
-spacing1 *size*	Specifies the space in screen units above first line of a paragraph with this tag.
-spacing2 *size*	Specifies the space in screen units between lines within a paragraph with this tag.
-spacing3 *size*	Specifies the space in screen units below the last line of a paragraph with this tag.
-tabs *tabList*	Specifies a list of tab stops consisting of offset values from the left edge in screen units followed by an optional justification. The first character in the text line must have the tag in order to take effect. Options are: **left** (default) with the left edge of text at tab position, **right** with text at tab position, **center** with the text centered at the tab position, or **numeric** with decimal point in the text is positioned at the tab position. If set to **{ }** the default 8-character tab stops are used.
-underline *boolean*	Specifies whether to underline text.
-wrap *mode*	Specifies how to wrap lines wider than the window. Options are: **char** (line break can be made after any character), **none** (no wrap), or **word** (line break can only be made at word boundaries).

Embedded Window Options

See Coordinates in Options and Resources for screen unit options.

Option	Description
-align *where*	Specifies window alignment if smaller than line height. Options are: **top** (align the top of window with the top of the text), **center** (center window within line), **bottom** (align the bottom of window with the bottom of the text), or **baseline** (align the bottom of window with the baseline of the text).
-create *script*	Specifies script to create and return window pathname if no **-window** option is given.
-padx *width*	Specifies extra space in screen units to leave on the left and right side of window.
-pady *height*	Specifies extra space in screen units to leave at the top and bottom of window.
-stretch *boolean*	Specifies whether window should be stretched vertically to fill line if less than the height of the line.
-window *pathName*	Specifies the name of window to display in the embedded window.

Embedded Image Options

See Coordinates in Options and Resources for screen unit options.

Option	Description
-align *where*	Specifies image alignment if smaller than line height. Options are: **top** (align the top of image with the top of the text), **center** (center image within line), **bottom** (align the bottom of image with the bottom of the text), or **baseline** (align the bottom of image with the baseline of the text).
-image *image*	Specifies the name of the image to embed. Returns error if *image* is not a valid image.
-name *imageName*	Specifies the name to use for referencing the embedded image. Appends *#nn* if *imageName* is already in use. Without **-name**, **-image** is used instead. Once an image is assigned a name, it cannot be changed with **image configure**.
-padx *width*	Specifies extra space in screen units to leave on the left and right side of the image.
-pady *height*	Specifies extra space in screen units to leave at the top and bottom of the image.

Selection Support

The **selectBackground**, **selectBorderWidth**, and **selectForeground** options for the text widget are tied to the **-background**,**-borderwidth**, and **-foreground** options for the sel tag. Changes in either will automatically be reflected in the other.

#	**Selection Criteria**
1	Whenever characters are tagged with **sel** the text widget will claim ownership of the selection.
2	Attempts to retrieve the selection will be serviced by the text widget, returning all the characters with the **sel** tag.
3	If the selection is claimed away by another application or by another window within this application, then the **sel** tag will be removed from all characters in the text.
4	(Tk 8.4+) Whenever the **sel** tag range changes a virtual event <<**Selection**>> is generated.

Undo Mechanism

(Tk 8.4+) If the **-undo** option is true, the text widget supports an unlimited undo and redo mechanism which records each insert and delete action in a stack. Boundaries (called "separators") are inserted between edit actions in order to group compound edits. An undo, uses all actions between separators then transfers them to the redo stack. The redo stack is cleared whenever new edit actions are recorded on the undo stack. Separators are inserted automatically when the **-autoseparators** option is true. The undo mechanism is also linked to the modified flag so undoing or redoing an edit can restore a text widget back to the unmodified or vice versa. Manual changes to the modified flag disable the automatic coupling until the flag has been reset.

Text Widget Commands

Command	Description
pathName **bbox** *index*	Returns a four element list with the upper left corner x and y coordinates, width, and height of the character or element at *index*. Only the visible portion will be returned or an empty list if not visible.
pathName **cget** *option*	Returns the current value of the configuration *option*. See Standard Options and Text Widget Specific Options above for *options*.
pathName **compare** *index1 op index2*	Compares the characters at indices *index1* and *index2* according to relational operator *op* and returns 1 if true, 0 if not. *Op* can be: <, <=, ==, >=, >, or !=.
pathName **configure** *?option? ?value? ?option value ...?*	Changes the configuration *option* to*value*. Without *value*, a list describing *option* is returned. Without *option*, a list of all available options for *pathName* is returned. For multiple options an empty string is returned. See Text Options above for *options*. Configuration options of each peer can be set independently except as indicated in the options above.
pathName count ?options? index1 index2	(Tk 8.5+) Counts the number of relevant things between the two indices and returns a list of integers based on options. If index1 is after index2, the result will be a negative number.
-chars	Count all characters, whether elided or not. Do not count embedded windows or images.
-displaychars	Count all non-elided characters.
-displayindices	Count all non-elided characters, windows and images.
-displaylines	Count all display lines from the line of the index1 up to, but not including the display line of index2.
-indices	Count all characters, embedded windows, and embedded images whether they are elided or not. Default option.
-lines	Count all logical lines (irrespective of wrapping) from the line of index1 up to, but not including the line of index2.
-update	Used before -ypixels to ensure that any possible out of date information is recalculated.
-xpixels	Count the number of horizontal pixels from the first pixel of index1 to (but not including) the first pixel of index2.
-ypixels	Count the number of vertical pixels from the first pixel of index1 to (but not including) the first pixel of index2.
pathName **debug** *?boolean?*	Specified whether internal consistency checks will be turned on for text widgets. In Tk 8.4+, global vars **tk_textRedraw** and **tk_textRelayout** are set to the indices that are redrawn. Without *boolean*, returns debugging status.
pathName **delete** *index1 ?index2 ...?*	Deletes contents of text widget from *index1* to just before *index2*, if specified, and *index2* > *index1* or just the character at *index1*. Newline characters can not be deleted. In Tk 8.4+, multiple ranges can be specified.
pathName **dlineinfo** *index*	Returns a five element list with the upper left corner x and y coordinates, width, height, and baseline in pixels of the display line containing *index*. Includes the portion of the line outside the window boundaries if no line wrap. If line the containing *index* is not visible, an empty list is returned.
pathName **dump** *?options? index1 ?index2?*	Returns contents of text widget from *index1* to just before *index2*, if specified, or just at *index1* in repeating *key value index* format. *Key* values are **text**,**mark**,**tagon**,**tagoff**, and **window**. **Value** is the text, mark name, tag name, or window name. *Index* is the start index of the text, mark, tag transition, or window. Options are:

-all	Return information about all elements
-command *command*	Invokes *command* with args *key*,*value*, and *index* for each text widget element within the range of indicies instead of returning it.
-image	(Tk 8.3.x+) Include image info in the dump results
-mark	Include mark info in the dump results.
-tag	Include tag transitions info (**tagon** and**tagoff**) in the dump results.
-text	Include text up to next element, newline, or *index2* in the dump results. Newlines are included in the dump.
-window	Include embedded windows info in the dump results. Returns window pathname or empty string if not created yet.
pathName **edit** *option ?arg ...?*	(Tk 8.4+) Controls the undo mechanism and the modified flag. Options are:
modified *?boolean?*	Sets the text widget modified flag. Without *boolean*, returns current state.
redo	If **-undo** is true, reapplies last undo edit if no edits have occurred since then. Generates error if redo stack is empty.
reset	Clears the undo and redo stacks.
separator	If **-undo** is true, inserts a separator (boundary) on the undo stack.
undo	If **-undo** is true, undoes last edit action (all insert, delete, etc. commands between two separators). Generates error if undo stack is empty.
pathName **get** ?options? ?--? *index1 ?index2 ...?*	Returns only characters from *index1* to just before *index2*, if specified, and *index2* > *index1*, or just at *index1*. An invalid range returns the empty string. In Tk 8.4+, multiple ranges can be specified and will be returned in the specified order. Options are:
-displaychars	(Tk 8.5+) Specifies that only those characters which are not elided will be returned.
pathName **image** *option ?arg ...?*	Controls embedded images. See Annotations above for more details on embedded images. Options are:
cget *index option*	Return current value of *option* for embedded image at *index*. For *options*, see Embedded Image Options above.
configure *index ?option value ...?*	Changes the embedded image configuration *option* to *value*. Without *value*, a list describing *option* is returned. Without *option*, a list of all available options for the image at *index* is returned. For multiple options an empty string is returned. For *options*, see Embedded Image Options above.
create *index ?option value ...?*	Create a new embedded image at position *index* with the specified options. For *options*, see Embedded Image Options above.
names	Returns a list of the names of all embedded images in the text widget.
pathName **index** index	Returns the position of *index* in *line.char* notation. See Indicies or Character Positions above.
pathName **insert** *index chars ?tagList? ?chars tagList ...?*	Inserts the char args just before the character at *index* using each tag in *tagList*. If *index* is at the end of the text (character after the last newline), then the new text is inserted just before the last newline instead. Without *tagList*, the new text will only use tags present in both the character at *index* and before *index*. Multiple *char tagList* args can be used.
pathName **mark** *option ?arg ...?*	Controls marks. See Annotations above for more details on marks. Options are:

gravity *markName ?direction?*	Specifies which adjacent character or direction (**left** or **right**) *markName* is attached to. Without *direction*, returns current gravity.
names	Returns a list of the names of all marks currently set.
next *index*	Returns name of next mark at or after *index* . Search starts at *index* unless its the name of a mark in which case it starts at the next mark. Returns empty string if no marks are left.
previous *index*	Returns name of previous mark at or before *index*. Search starts at the character before *index* unless its the name of a mark in which case it starts before the mark. Returns empty string if no marks are left.
set *markName index*	Creates mark *markName* or moves it if it already exists to just before the character at *index*.
unset *markName ?markName ...?*	Removes each specified mark so they are no longer usable as indices.
pathName peeroption ?args?	(Tk 8.5+) Used to create and query widget peers. A peer widget has complete access to pathName widget's data while maintaining separate config options except as noted in the config options above.
create newPathName ?options?	Creates a peer text widget with the given newPathName, and any specified config options. By default the peer will have the same start and end line as the parent widget.
names	Returns a list of peers of this widget excluding this widget.
pathName replace index1 index2 chars ?tagList? ?chars tagList ...?	(Tk 8.5+) Replaces the range of characters from index1 to just before index2 with the given characters and tags in tagList. Without *tagList*, the new text will only use tags present in both the character at *index1* and *index2*.
pathName **scan** *option args*	Controls scanning on text widgets. *Options* are:
mark *x y*	Records *x* and *y* and the current view in the text widget. Typically associated with mouse button press in widget at coordinates *x*,*y*.
dragto *x y*	Adjusts the view by 10 times the difference between the coordinates *x*,*y* and the last **mark** *x ,y* coordinates. Used with mouse motion events to produce high speed dragging effect.
pathName **search** *?switches? ?--? pattern index ?stopIndex?*	Searches for a match to *pattern* in the range of text from *index* to *stopIndex*, if specified, or back to *index* and returns the index of the match. Without stopIndex, the search wraps around at the end/beginning of the text. The matching range must be entirely within a single line of text. Switches are:
-all	(Tk 8.5+) Find all matches in the given range and return a list of the indices of the first character of each match.
-backwards	Search backwards in the text from *index* .
-count *varName*	Stores the length of the matched text and elements in *varName*. Used with -all, returns a list of counts.
-elide	(Tk 8.3+) Find elidden (hidden) text as well. By default only displayed text is searched.
-exact	The characters must exactly match *pattern* . Newlines are not removed from the line end before checking for a match. (Default)
-forwards	Search forward in the text from *index*. (Default)
-nocase	Ignore case differences between *pattern* and the text.
-nolinestop	(Tk 8.5+) Used with **-regexp** to allow . and [^ sequences to match the newline character \n.
-overlap	(Tk 8.5+) Used with -all, so that all matches which are not totally enclosed within another match are returned. Default is that matches which overlap an already-found match will not be returned.
-regexp	Use Regular Expression *pattern* matching. Newlines are removed from the line end before checking for a match.
pathName **see** *index*	Adjust the view in window so character at *index* is completely visible. For small adjustments the text is scrolled just enough to see the text. For large adjustments, the text is centered in the window.
pathName **tag** *option ?arg ...?*	Controls tags. See Annotations above for more details on tags. Options are:

add *tagName index1 ?index2 index1 index2 ...?*	Apply tag *tagName* to characters in given range from *index1* to just before *index2*. Multiple ranges are supported.
bind *tagName ?sequence? ?command?*	Create a binding to evaluate *command* whenever event in *sequence* occurs within the characters or elements associated with *tagName*. See **bind** command for options. Only mouse, keyboard, and virtual events can be used. An **Enter** event for a tag triggers when the tag first becomes present on the current character, and a **Leave** event triggers when it ceases to be present on the current character. **Enter** and **Leave** events can happen either because the current mark moved or because the character at that position changed. When a character has multiple tags with bindings, only one binding is invoked for each tag in lowest to highest priority order. If there are multiple bindings for a tag, the most specific binding is used. If bindings exist for the parent widget, they will be invoked after the tag bindings.
cget *tagName option*	Return current value of *option* for tag*tagName*. For *options*, see Tag Options above.
configure *tagName ?option? ?value? ?option value ...?*	Changes the tag *tagName* configuration *option* to *value*. Without *value*, a list describing *option* is returned. Without *option*, a list of all available options for the image at *index* is returned. For multiple options an empty string is returned. For *options*, see Tag Options above.
delete *tagName ?tagName ...?*	Delete all tag information (tags from characters, bindings, etc.) for each *tagName* arg.
lower *tagName ?belowThis?*	Change priority of tag *tagName* so it is just below tag *belowThis* or without *belowThis* below all tags.
names *?index?*	Returns a list of the names of all tags associated with character at *index* in increasing prioity order. Without *index*,a list of all defined tags is returned.
nextrange *tagName index1 ?index2?*	Searches between *index1* to just before*index2* (default is end of text) for the first region tagged with *tagName*. Returns a two element list with the character range (start and end+1) of region found or empty string if none.
prevrange *tagName index1 ?index2?*	Searches between just before *index1* to*index2* (default is index 1.0) for the first region tagged with *tagName* . Returns a two element list with the character range (start and end+1) of region found or empty string if none.
raise *tagName ?aboveThis?*	Change priority of tag *tagName* so it is just above tag *aboveThis* or without *aboveThis* above all tags.
ranges *tagName*	Returns a list describing all character ranges tagged with *tagName*. Each pair of elements contains the start and end+1 index for that range. If no matches are found an empty list is returned.
remove *tagName index1 ?index2 index1 index2 ...?*	Remove tag *tagName* from all characters in given range from *index1* to just before *index2*, if specified, and *index2* > *index1*, or just at *index1*. Multiple ranges are supported.
pathName **window** *option ?arg ...?*	Controls embedded windows. See Annotations above for more details on embedded windows. Options are:
cget *index option*	Return current value of *option* for embedded window at *index*. For *options*, see Embedded Window Options above.
configure *index ?option value ...?*	Changes the embedded window configuration *option* to*value*. Without *value*, a list describing *option* is returned. Without *option*, a list of all available options for the image at *index* is returned. For multiple options an empty string is returned. For *options*, see Embedded Window Options above.
create *index ?option value ...?*	Create a new embedded window at position *index* with the specified options. For *options*, see Embedded Window Options above.
names	Returns a list of the names of all embedded windows in the text widget.
pathName **xview** *?option args*?	Query or change the horizontal text widget view. Without any *options*, a two element list is returned specifying the start and end of the visible fraction (from 0 to 1) of the horizontal span of the widget between the left and right edges of the window. Vaild *options* and *args* are:

moveto *fraction*	Adjust the view in the window so that *fraction* (from 0 to 1) of the total width of the widget is off-screen to the left.
scroll *number* **pages**	Shift the view left (number < 0) or right (number > 0) by number screenfuls.
scroll *number* **units**	Shift the view left (number < 0) or right (number > 0) by average-width number of characters on the display.
pathName **yview** *?option args*?	Query or change the vertical text widget view. Without any *options*, a two element list is returned specifying the start and end of the visible fraction (from 0 to 1) of the vertical span of the widget between the top and bottom edges of the window. Vaild *options* and *args* are:
moveto *fraction*	Adjust the view in the window so that *fraction* (from 0 to 1) of the total height of the widget is off-screen to the top.
scroll *number* **pages**	Shift the view up (number < 0) or down (number > 0) by number screenfuls.
scroll *number* **units**	Shift the view up (number < 0) or down (number > 0) by number lines.
?**-pickplace**?*index*	(Obsolete) Changes the view in the widget's window to make index visible using the following criteria: if *index* is already visible then don't do anything, if *index* is a few lines off-screen above the window position it at the top of the window, if *index* is a few lines off-screen below the window position it at the bottom of the window, otherwise center *index* in the window.
number	(Obsolete) Makes the first character on the line after the one given by *number* visible at the top of the window.

Default Text Widget Bindings

For additional default bindings see Virtual Events in Bindings and Virtual Events.

Event	Description
<Button-1>	Positions the insertion cursor just before the character underneath the mouse cursor, sets the input focus to this widget, and clears any selection in the widget.
<B1-Motion>	Drags out a selection (in words if double clicked or lines if triple clicked) between the insertion cursor and the character under the mouse.
<Double-Button-1>	Selects the word under the mouse and positions the insertion cursor at the beginning of the word.
<Triple-Button-1>	Selects all of the text on the line and positions the insertion cursor before the first character.
<Shift-B1-Motion>	Adjusts the end of the selection (in words if double clicked or lines if triple clicked) that was nearest to the mouse cursor when button 1 was pressed.
<B1-Leave>	Adjusts the view in the widget in the direction that the mouse left the window more quickly.
<B1-Enter> or **<ButtonRelease-1>**	Stops cancel repeat.
<Control-Button-1>	Position the insertion cursor in the widget without affecting the selection.
<Button-2>	Paste selection into the widget at the position of the mouse cursor.
<B2-Motion>	Adjusts view in widget by scrolling in the direction of the mouse movement.
<MouseWheel> or **<B4>** and **<B5>**	(Tk 8.0.4+) Adjusts view in widget up or down in increments of 4 lines.
<Left> or **<Control-b>**	Moves the insertion cursor one character back (left), clears any selection, and sets the selection anchor.
<Right> or **<Control-f>**	Moves the insertion cursor one character forward (right), clears any selection, and sets the selection anchor.
<Shift-Left>	Move the insertion cursor one character back (left) and extend the selection to include the new character.
<Shift-Right>	Move the insertion cursor one character forward (right) and extend the selection to include the new character.
<Control-Left> or **<Meta-b>**	Move the insertion cursor back (left) by one word, clears any selection, and sets the selection anchor.
<Control-Right> or **<Meta-f>**	Move the insertion cursor forward (right) by one word, clears any selection, and sets the selection anchor.

<Shift-Control-Left>	Move the insertion cursor back (left) by one word and also extend the selection.
<Shift-Control-Right>	Move the insertion cursor forward (right) by one word and also extend the selection.
<Up> or **<Control-p>**	Moves the insertion cursor up one line, clears any selection, and sets the selection anchor.
<Down> or **<Control-n>**	Moves the insertion cursor down one line, clears any selection, and sets the selection anchor.
<Shift-Up>	Move the insertion cursor up one line and extend the selection to include the new line.
<Shift-Down>	Move the insertion cursor down one line and extend the selection to include the new line.
<Control-Up>	Move the insertion cursor up by one paragraph, clears any selection, and sets the selection anchor.
<Control-Down>	Move the insertion cursor down by one paragraph, clears any selection, and sets the selection anchor.
<Shift-Control-Up>	Move the insertion cursor up by one paragraph and also extend the selection.
<Shift-Control-Down>	Move the insertion cursor down by one paragraph and also extend the selection.
<Prior>	Moves the insertion cursor up one screenful, clears any selection, and sets the selection anchor.
<Next>	Moves the insertion cursor down one screenful, clears any selection, and sets the selection anchor.
<Shift-Prior>	Move the insertion cursor up by one screenful and also extend the selection.
<Shift-Next>	Move the insertion cursor down by one screenful and also extend the selection.
<Control-v>	(MS Windows only) Adjusts view in widget down by one screenful without moving the insertion cursor or adjusting the selection.
<Control-Prior>	Adjusts view in widget left by one screenful without moving the insertion cursor or adjusting the selection.
<Control-Next>	Adjusts view in widget right by one screenful without moving the insertion cursor or adjusting the selection.
<Home> or **<Control-a>**	Move the insertion cursor to the beginning of the line and clears any selection.
<Shift-Home>	Move the insertion cursor to the beginning of the line and also extends the selection to that point.
<End> or **<Control-e>**	Move the insertion cursor to the end of the line and clears any selection.
<Shift-End>	Move the insertion cursor to the end of the line and also extends the selection to that point.
<Control-Home> or **<Meta-less>**	Move the insertion cursor to the beginning of the text and clears any selection.
<Shift-Control-Home>	Move the insertion cursor to the beginning of the text and also extends the selection to that point.
<Control-End> or **<Meta-greater>**	Move the insertion cursor to the end of the text and clears any selection.
<Shift-Control-End>	Move the insertion cursor to the end of the text and also extends the selection to that point.
<Tab>	Insert tab character and sets focus to current window.
<Shift-Tab>	No function.
<Control-Tab>	Changes focus to next window.
<Control-Shift-Tab>	Changes focus to previous window.
<Control-i>	Insert tab character.
<Return>	Insert newline character and add separator to undo stack.
<Select> or **<Control-Space>**	Set the selection anchor to the position of the insertion cursor without affecting the selection.

<Shift-Select> or **<Shift-Control-Space>**	Adjusts the selection to the current position of the insertion cursor, if there is one, otherwise it selects from the anchor to the insertion cursor.
<Control-slash>	Selects all the text in widget.
<Control-backslash>	Clears any selection in the widget.
<Delete>	Deletes the selection, if there is one, otherwise it deletes the character to the right of the insertion cursor.
<BackSpace> or **<Control-h>**	Deletes the selection, if there is one, otherwise it deletes the character to the left of the insertion cursor.
<Insert>	Insert current selection from clipboard at insertion cursor position.
<Control-x>	Deletes the selection in the widget.
<Control-d>	Deletes the character to the right of the insertion cursor.
<Control-k>	Deletes all the characters right of the insertion cursor to the end of the line. If insertion cursor is at the end of the line then the newline is deleted.
<Control-t>	Reverses (transposes) the order of the two characters to the right of the insertion cursor.
<Control-o>	Opens a new line by inserting a newline character in front of the insertion cursor without moving the insertion cursor.
<Meta-d>	Deletes the word to the right of the insertion cursor.
<Meta-BackSpace> or **<Meta-Delete>**	Deletes the word to the left of the insertion cursor.
<Keypress>	Insert character into widget.
<<Undo>> or **<Control-z>** or **<Control-underscore>**	(Tk 8.4+) Perform **edit undo** if the -undo option is true.
<<Redo>> or **<Control-Z>** or **<Control-y>**	(Tk 8.4+) Perform **edit redo** if the -undo option is true. (MS Windows only **<Control-y>**)
<<Selection>>	(Tk 8.4+) Generated whenever the**sel** tag range changes.
<<Modified>>	(Tk 8.4+) Generated whenever the text widget modified flag changes state.
<<Paste>>	Paste the contents of the clipboard into the text widget.

3.26 Toplevel Window

Command	Description
toplevel *pathName ?options?*	Creates a toplevel window *pathName* with *options* and returns the new widget's path name. Used as a container for other widgets.

Toplevel Window Options

Standard

See Common Options and Resources in Options and Resources for full details.

-borderwidth	**-highlightcolor**	**-pady** (Tk 8.4+)
-cursor	**-highlightthickness**	**-relief**
-highlightbackground	**-padx** (Tk 8.4+)	**-takefocus**

Toplevel Window Specific

See Coordinates in Options and Resources for screen unit options.

Configure Option	Resource Name	Resource Class	Description
-background *color*	**background**	**Background**	Same as standard **-background** expect if set to empty string, the widget will not display or allocate a colormap entry for the background or border color.
-class *name*	**class**	**Class**	Specifies class name to use in querying the option database and for bindings. Can not be changed with **configure** command.
-colormap *colormap*	**colormap**	**Colormap**	Specifies colormap (default is same as parent) to use for the window where *colormap* can be **new** (allocate new colormap) or the name of another window on same display with same visual. Can not be changed with **configure** command.
-container *boolean*	**container**	**Container**	Specifies whether the toplevel will be a container to embed another application. Can not be changed with **configure** command.
-height *height*	**height**	**Height**	Height of toplevel window in screen units.
-menu *pathName*	**menu**	**Menu**	Specifies the menu widget to be used as a menubar at the top of the window (or screen for Macs).
-screen *screen*			Screen on which to place the window.
-use *windowID*	**use**	**Use**	Toplevel should be embedded inside window identified by *windowID* (see **winfo id**) which was created as a container.
-visual *visual*	**visual**	**Visual**	Specifies the visual to use for the window. Default is the same as the parent. See Screen or Window Visuals below for options. Can not be changed with **configure** command.
-width *width*	**width**	**Width**	Width of toplevel window in screen units.

Toplevel Window Commands

Command	Description
pathName **cget** *option*	Returns the current value of the configuration *option*. See Toplevel Window Options above for *options* .
pathName **configure** *?option? ?value? ?option value ...?*	Change the configuration *option* to *value*. Without *value*, a list describing *option* is returned. Without *option*, a list of all available options for *pathName* is returned. For multiple options an empty string is returned. See Top Level Options above for *options*.

Screen or Window Visuals

Visual	Description
class depth	Class name followed by integer *depth*. Classes are: **directcolor**, **grayscale**, **greyscale**, **pseudocolor**, **staticcolor**, **staticgray**,**staticgrey**, or **truecolor**. *Depth* specifies the bits per pixel for the visual. Same logic as **best** option (a).
default	Use the default visual for current screen.
pathName	Use same visual as window *pathName* . Must be on the same screen.
number	Use the visual whose X identifier is *number*.
best *?depth?*	Choose the "best possible" visual, in decreasing order of priority: (a) visual decreasing order: visual with exact *depth*, visual with depth > *depth* (but as little extra as possible), visual with depth < *depth*(but with the greatest depth possible); (b) without *depth*, then the deepest available visual is used; (c) class in decreasing order: **pseudocolor**,**truecolor**, or **directcolor**, **staticcolor**, **staticgray**, or **grayscale** ; (d) the default visual for the screen is better than any other visual.

3.27 Window Information

See Coordinates in Options and Resources for screen unit options.

Command	Description
winfo atom ?**-displayof** *window? name*	Returns integer identifier for the atom given by *name* on *window*'s display (default is the same as application's main window).
winfo atomname ?**-displayof** *window? id*	Returns textual name of the atom given by integer *id* on *window*'s display (default is the same as application's main window).
winfo cells *window*	Returns the number of cells in the colormap for *window*.
winfo children *window*	Returns a list containing the path names of *window*'s children in stacking order except for top-level windows.
winfo class *window*	Returns the class name of *window*.
winfo colormapfull *window*	Return 1 if the colormap for *window* is full, 0 if not.
winfo containing ?**-displayof** *window? rootX rootY*	Returns the path name of window highest in the stacking order containing the point given by *rootX* and *rootY* (in screen units) on *window*'s display (default is the same as application's main window) or empty string if none.
winfo depth *window*	Returns the depth of *window* in bits per pixel.
winfo exists *window*	Returns 1 if *window* exists, 0 if not.
winfo fpixels *window number*	Returns a floating-point value giving the number of pixels in *window* corresponding to *number* distance in screen units.
winfo geometry *window*	Returns the pixel geometry for *window*, in the form *width*x*height*Â±*x*Â±*y*.
winfo height *window*	Returns height of *window* in pixels.
winfo id *window*	Returns a hexadecimal string indicating the platform-specific identifier for *window*.
winfo interps ?**-displayof** *window?*	Returns a list of all Tcl interpreters registered on *window*'s display (default is the same as application's main window).
winfo ismapped *window*	Returns 1 if *window* is currently mapped, 0 if not.
winfo manager *window*	Returns the name of the geometry manager currently responsible for *window* or empty string if none.
winfo name *window*	Returns *window*'s name within its parent, as opposed to its full path name.
winfo parent *window*	Returns the path name of *window*'s parent or empty string if *window* is the main window of the application.
winfo pathname ?**-displayof** *window? id*	Returns the path name of the window whose X identifier is *id* on *window*'s display (default is the same as application's main window).
winfo pixels *window number*	Returns the number of pixels in *window* rounded to the nearest integer value corresponding to *number* distance in screen units.
winfo pointerx *window*	Returns the mouse pointer's x root coordinate in pixels on *window*'s screen. Returns -1 if the mouse pointer isn't on the same screen as *window*.
winfo pointerxy *window*	Returns a two element list of mouse pointer's x and y root coordinates in pixels on *window*'s screen. Returns -1 for each coordinate if the mouse pointer isn't on the same screen as *window*.
winfo pointery *window*	Returns the mouse pointer's y root coordinate in pixels on *window*'s screen. Returns -1 if the mouse pointer isn't on the same screen as *window*.
winfo reqheight *window*	Returns a decimal string giving *window*'s requested height, in pixels.
winfo reqwidth *window*	Returns a decimal string giving *window*'s requested width, in pixels.

winfo rgb *window color*	Returns a list of the three RGB values that correspond to *color* in *window*. See Colors in Options and Resources for valid *color* formats.
winfo rootx *window*	Returns the x-coordinate of the upper-left corner of *window* (including its border if present) in the root window of the screen.
winfo rooty *window*	Returns the y-coordinate of the upper-left corner of *window* (including its border if present) in the root window of the screen.
winfo screen *window*	Returns the name of the screen associated with *window*, in the form *displayName.screenIndex*.
winfo screencells *window*	Returns the number of cells in the default color map for *window*'s screen.
winfo screendepth *window*	Returns the depth of *window*'s screen in bits per pixel.
winfo screenheight *window*	Returns the height of *window*'s screen in pixels.
winfo screenmmheight *window*	Returns the height of *window*'s screen in millimeters.
winfo screenmmwidth *window*	Returns the width of *window*'s screen in millimeters.
winfo screenvisual *window*	Returns the visual class of *window*'s screen. Options are: **directcolor**, **grayscale**, **pseudocolor**,**staticcolor**, **staticgray**, or **truecolor**.
winfo screenwidth *window*	Returns the width of *window*'s screen in pixels.
winfo server *window*	Returns a platform specific formatted string containing information about the server for *window*'s display.
winfo toplevel *window*	Returns the pathname of the top-level window containing *window*.
winfo viewable *window*	Returns 1 if *window* and all of its ancestors up through the nearest toplevel window are mapped, 0 if not.
winfo visual *window*	Returns the visual class of *window*. Options are: **directcolor**, **grayscale**, **pseudocolor**,**staticcolor** , **staticgray**, or **truecolor**.
winfo visualid *window*	Returns the X identifier of the visual for *window*.
winfo visualsavailable *?window***includeids**?	Returns a list that describes the visuals available for *window*'s screen where each element is a sublist of the class and depth in bits per pixel.
winfo vrootheight *window*	Returns the height of the virtual root window associated with *window* if there is one, otherwise the height of *window* 's screen.
winfo vrootwidth *window*	Returns the width of the virtual root window associated with *window* if there is one, otherwise the width of *window*'s screen.
winfo vrootx *window*	Returns the x-offset of the virtual root window associated with *window* relative to the root window of its screen or 0 if there isn't a virtual root window for *window*.
winfo vrooty *window*	Returns the y-offset of the virtual root window associated with *window* relative to the root window of its screen or 0 if there isn't a virtual root window for *window*.
winfo width *window*	Returns *window*'s width in pixels.
winfo x *window*	Returns x-coordinate of the upper-left corner of *window* (including its border if present) in its parent.
winfo y *window*	Returns y-coordinate of the upper-left corner of *window* (including its border if present) in its parent.

3.28 Window Management

Command	Description
bell ?**-displayof** *window*? ?**-nice**?	Rings the X bell on *window's* (default is ".") display. In Tk 8.4+ the screen saver is reset if **-nice** is not specified.

destroy *?window ...?*	Deletes each *window* and all their descendents. If window "." is destroyed, the entire application will be deleted. Windows are deleted in order, but stops if an error occurs. If a window doesn't exist, no error is returned.
focus	Returns the path name of the focus window on the display containing the application's main window or an empty string if not the same application.
focus *window*	Changes focus to *window* on *window*'s display. Does not alter which top-level has the input focus for the the display.
focus -displayof *window*	Returns the path name of the focus window on the display containing *window* or an empty string if not the same application.
focus -force *window*	Sets the focus of *window*'s display to *window*, even if the application doesn't currently have the input focus for the display.
focus -lastfor *window*	Returns the name of the most recent window to have the input focus among all the windows in the same top-level as *window*. If none or it was deleted, then the name of the top-level is returned.
grab *?***-global***? window*	Same as **grab set**.
grab current *?window?*	Returns name of current grab window on *window*'s display or empty string if not the same application. Without *window*, returns list of all windows grabbed by application for all displays.
grab release *window*	Releases grab on *window*.
grab set *?***-global***? window*	Sets a local grab (grabbing application only) on *window* unless **-global** (locks out all other apps on screen except subtree of grabbing app) is specified. If grab was already in effect, it is released.
grab status *window*	Returns current grab state (**none**,**local**, or **global**) for *window*.
lower *window ?belowThis?*	Places *window* below window *belowThis* (default is below all siblings) in stacking order.
raise *window ?aboveThis?*	Places window above window *aboveThis* (default is above all siblings) in stacking order. In Tk 8.3.4+, it does not block for 2 seconds.
tk_focusFollowsMouse	Change focus model of application to an implicit one where the focus follows the mouse pointer.
tk_focusNext *window*	Returns the window just after *window* in focus order.
tk_focusPrev *window*	Returns the window just before *window* in focus order.
tk appname *?newName?*	Sets and returns the application's interpreter name to *newName* (must not start with uppercase char) appending # and an integer if necessary to create a unique name. Without *newName* , returns current name. Reenables the **send** command if it was deleted.
tk caret *window* ?**-x** *x*? ?**-y** *y*? ?**-height** *height*?	(Tk 8.4+) Sets the caret location for the display of the specified Tk window *window*. The caret is the per-display cursor location used for indicating global focus. *x* and *y* represent window-relative coordinates, and *height* is the height of the current cursor location, or the height of the specified *window* if none is given. Without any options, the last used values are returned.
tk scaling ?**-displayof** *window? number*	Set scaling factor for conversion between physical units and pixels on *window*'s display (default is current) where *number* (floating point value) is the pixels per point (1/72 inch).
tk useinputmethods ?**-displayof** *window? ?boolean?*	(Tk 8.3+) Specifies whether Tk should use XIM (X Input Methods) for filtering events on *window* (default is main window). Without *boolean*, the current setting is returned (default is on in Tk 8.3.3+ and off for previous versions).
tk windowingsystem	(Tk 8.4+) Returns the current Tk windowing system. Options are: **x11** (X11-based), **win32** (MS Windows), **classic** (Mac OS Classic), or **aqua** (Mac OS X Aqua).
wm aspect *window ?minNumer minDenom maxNumer maxDenom?*	Specifies the aspect ratio of *window* (width/length) to be constrained to lie between *minNumer*/*minDenom* and *maxNumer* /*maxDenom*. If all are set to empty strings, then any existing aspect ratio restrictions are removed. Without options a list of the current values is returned.
wm attributes *window ?option value ...?*	(Tk 8.4+) Change the platform specific window manager attribute (used by MS Windows only) *option* to *value* . Without *value*, the current value for *option* is returned. Without *option*, a list of all platform specific flags and their values is returned. Options are:

-alpha value	(Tk 8.4.8+ MS Windows, Mac OSX) set alpha transparency. 0.0 (completely transparent) to 1.0 (opaque). Default is 1.0.
-disabled ?boolean?	(MS Windows only) Window disabled status
-fullscreen ?boolean?	(Tk 8.5+ MS Windows only) Requests that the window should fill the entire screen and have no window decorations.
-modified ?boolean?	(Mac OSX) Window modification state (determines whether the window close widget contains the modification indicator).
-titlepath ?path?	(Mac OSX) Window proxy title path (file referenced as the window proxy icon which can be dragged and dropped in lieu of the file's finder icon).
-toolwindow ?boolean?	(MS Windows only) Specifies style of the window as toolwindow.
-topmost ?boolean?	(MS Windows only) Requests that this window should be kept above all other windows that do not also have the -topmost attribute set.
-zoomed?boolean?	(UNIX TBD) Requests that the window should be maximized. Same as wm state zoomed on Windows.
wm client *window ?name?*	Store *name* in *window*'s **WM_CLIENT_MACHINE** property to specify the machine the window is running on. Without *name*, returns last name set with **wm client**. If set to an empty string, the property is deleted.
wm colormapwindows *window ?windowList?*	Store *windowList* in *window*'s **WM_COLORMAP_WINDOWS** property to identify the internal windows within *window* that have private colormaps. Without *windowList*, returns a list of windows in the property with different colormaps.
wm command *window ?value?*	Store list *value* in *window*'s **WM_COMMAND** property. Informs window manager of command used to invoke the application. Without *value*, returns last value set with **wm command**. If set to an empty string, the property is deleted.
wm deiconify *window*	Arrange for *window* to be displayed (mapped) in normal (non-iconified) form.
wm focusmodel *window ?option?*	Specifies the focus model for window. Options are: **active** (claim focus for itself or decendents) or **passive** (default option to never claim focus for itself).Without *option*, returns the current focus model.
wm frame *window*	Returns the platform specific window identifier for the outermost decorative frame that contains *window*. If *window* has none, returns the platform specific ID of window itself.
wm geometry *window ?newGeometry?*	Changes geometry of *window* to *newGeometry* using in the form of: *width*x*height*Â±*x*Â±*y*. Without *newGeometry*, returns current geometry.
wm grid *window ?baseWidth baseHeight widthInc heightInc?*	Indicates that window is to be managed as a gridded window with the specified relation between grid and pixel units. *BaseWidth* and *baseHeight* specify the number of grid units corresponding to the pixel dimensions requested internally by *window*.*WidthInc* and *heightInc* specify the number of pixels in each horizontal and vertical grid unit. If all are set to empty strings, then *window* will no longer be managed as a gridded window. Without options a list of the current values is returned.
wm group *window ?pathName?*	Gives path name for leader of group to which*window* belongs. Without *pathName*, returns *window* 's current group leader. When set to empty string, *window* is removed from any groups.
wm iconbitmap *window ?bitmap?*	Specifies a bitmap to use as icon image when *window* is iconified. If set to an empty string, then the current bitmap is cancelled. Without *bitmap*, the current bitmap name is returned or empty string if none. If a "@" is the first char, the bitmap is a filename. Unix uses .xbm files and windows uses .ico files.
wm iconbitmap *window* -default filename.ico	(Tk 8.3.3+, MS Windows only) Specifies a bitmap file to use as icon image when *window* is iconified. Overridden by wm iconphoto.
wm iconify *window*	Arrange for *window* to be iconified.
wm iconmask *window ?bitmap?*	Specifies a bitmap to use to mask icon image when *window* is iconified. If set to an empty string, then the current bitmap is cancelled. Without *bitmap*, the current bitmap name is returned or empty string if none.

wm iconname *window ?newName?*	Specifies name to use as a label for *window* 's icon. If set to an empty string, then the current name is cancelled and the window's title is used. Without *newName*, the current name is returned or empty string if none.
wm iconphoto window ?-default? image1 ?image2 ...?	(Tk 8.5+ MS Windows and UNIX) Specifies a image to use as icon image in titlebar and when *window* is iconified. If **-**default is specified, this is applied to all future created toplevels as well. Multiple images are accepted to allow different images sizes (eg, 16x16 and 32x32) to be provided. On UNIX, only use 2 images and put larger image first.
wm iconposition *window ?x y?*	Specifies hints for position *x* and *y* on root window to place *window*'s icon. If set to empty strings, then the current position is cancelled. Without *x y*, a list of the current values is returned or empty string if none.
wm iconwindow *window ?pathName?*	Specifies the path name of window to use as the icon when *window* is iconified. If set to an empty string, then the current icon window is cancelled. Without *pathName*, the current name of the icon window is returned or empty string if none.
wm maxsize *window ?width height?*	Specifies maximum window size for *window* in pixels or grids for gridded windows. If set to empty strings, the sizes default to the screen size. Without *width height*, a list of the current max sizes is returned.
wm minsize *window ?width height?*	Specifies mimum window size for *window* in pixels or grids for gridded windows. If set to empty strings, the sizes default to one pixel in each dimension. Without *width height*, a list of the current min sizes is returned.
wm overrideredirect *window ?boolean?*	Specifies the override-redirect flag for *window* which is commonly used by the window manager to determine whether window should show a decorative frame.
wm positionfrom *window ?who?*	Specifies whether window's current position was **program** or **user** requested. If set to an empty string, the current position source is cancelled. Without *who*, the current position source is returned.
wm protocol *window ?name? ?command?*	Specify a Tcl command to be invoked for messages of protocol *name*. Valid values for *name* are: **WM_DELETE_WINDOW**, **WM_SAVE_YOURSELF**, or **WM_TAKE_FOCUS**. Without *command* , the current command for *name* is returned. If *name* is set to an empty string, then current handler is deleted. Without *name or command* , a list of all protocol handlers is returned.
wm resizable *window ?widthBoolean heightBoolean?*	Specifies whether *window*'s width and/or height is resizable (default is true for both). Without the options, a list of the current values is returned.
wm sizefrom *window ?who?*	Specifies whether window's current size was **program** or **user** requested. If set to an empty string, the current size source is cancelled. Without *who*, the current size source is returned.
wm stackorder *window ?option? ?newWindow?*	(Tk 8.4+) Returns stacking order of *window*'s children in lowest to highest order. Returns relative position of *window* compared to *newWindow* based on options **isabove** and **isbelow**.
wm state *window ?newState?*	Returns current state of window. In Tk 8.3+, *newState* changes the current state of window. Options are: **normal**, **icon**, **iconic**, **withdrawn**, and **zoomed** (MS Windows only).
wm title *window ?string?*	Specifies the title for *window*'s decorative frame. Without *string*, the current name is returned.
wm transient *window ?master?*	Informs window manager that *window* is a transient of the window *master*. If set to an empty string, then *window* is not marked as a transient window. Without *master*, the path name of *window*'s current master, or an empty string if none, is returned.
wm withdraw *window*	Arranges for *window* to be withdrawn (unmapped) from the screen.

4 Other Tcl Packages

4.1 dde Package

(TCL 8.1+, MS Windows only) Execute a Dynamic Data Exchange (DDE) command. DDE is used by windows applications to exchange data. Each DDE transaction needs a service name and a topic. Tcl uses the service name TclEval, while the topic name is the name of the interpreter given by dde servername.

Command	Description
dde eval ?-async? topic cmd ?arg arg ...?	(Tcl 8.5+) Evaluates a command and its arguments using the interpreter specified by topic. The DDE service must be the TclEval service. The -async option requests asynchronous invocation. Returns an error message if the script did not run unless the -async option was specified.
dde execute ?-async? service topic data	Sends data to the server indicated by service with the topic topic. Typically the application name is the service, the filename is the topic, and data is a script to be run on the file. The -async flag requests an asynchronous invocation. An error message will be returned if the script does not run unless the -async flag was specified.
dde poke service topic item data	(Tcl 8.2+) Sends data as the value for item to the server indicated by service with the topic topic. Typically the application name is the service, the command or filename is the topic, item is application specific but is often not used (can't be null), and data is the value to use.
dde request ?-binary? service topic item	Returns the value of item from the server indicated by service with the topic topic. Typically the application name is the service, the filename is the topic, and item is application specific. In Tcl 8.4+, if -binary is specified, the result is returned as a byte array, otherwise a null terminated string is assumed.
dde servername ?options? ?--? ?topic?	Registers the interpreter as a DDE server with the service name TclEval and the topic name topic. Without topic, the current topic or an empty string (if no service is registered) will be returned.
-force	(Tcl 8.5+) Forces registration of precisely the given topic name.
-handler proc	(Tcl 8.5+) Specifies a Tcl procedure that will be called to process calls to the dde server. Must be used if the package has been loaded into a safe interpreter. The procedure is called with all the arguments provided by the remote call.
dde services service topic	Returns a list of service-topic pairs that currently exist on the machine matching service and topic. If a null is used for service and/or topic, all services and/or topics will be returned. Returns a null if no matches were found.

4.2 http Package

(Tcl 8.0+)

Command	Description
::http::cleanup token	
::http::code token	
::http::config ?options?	
-accept mimetypes	
-proxyhost hostname	
-proxyport number	
-proxyfilter command	
-urlencoding encoding	(Tcl 8.4.7+)
-useragent string	
::http::data token	
::http::error token	
::http::formatQuery key value ?key value ...?	
::http::geturl url ?options?	
-binary boolean	
-blocksize size	
-channel name	
-command callback	
-handler callback	
-headers keyvaluelist	
-progress callback	
-query query	
-queryblocksize size	
-querychannel channelID	
-queryprogress callback	
-timeout milliseconds	
-type mime-type	(Tcl 8.2.3+)
-validate boolean	
::http::ncode token	
::http::register proto port command	(Tcl 8.2.3+)
::http::reset token ?why?	
::http::size token	
::http::status token	
::http::unregister proto	(Tcl 8.2.3+)
::http::wait token	

4.3 msgcat Package

The msgcat package provides a set of functions that can be used to manage multi-lingual user interfaces. Text strings are defined in a ‘‘message catalog’’ which is independent from the application, and which can be edited or localized without modifying the application source code. New languages or locales are provided by adding a new file to the message catalog.

Command	Description
::msgcat::mc src-string ?arg arg ...?	(Tcl 8.1+) Returns a translation of src-string according to the user's current locale. Searches from the current namespace up to the global namespace. If none found, returns result of ::msgcat::mcunknown. If additional arguments past src-string are given, the format command is used to substitute the additional arguments in the translation of src-string.
::msgcat::mcload dirname	(Tcl 8.4+) Reads the contents of files in dirname that match the language specifications returned by ::msgcat::mcpreferences and have a ".msg" extension.
::msgcat::mclocale ?newLocale?	(Tcl 8.1+) Sets the locale to newLocale (case insensitive). Without newLocale, returns the current locale.
::msgcat::mcmax ?src-string src-string ...?	(Tcl 8.4+) Returns the length of the longest translated src-string.
::msgcat::mcmset locale src-trans-list	(Tcl 8.4+) Sets the translation for multiple source strings in src-trans-list (list of src-string and translate-string pairs) in the specified locale and the current namespace.
::msgcat::mcpreferences	(Tcl 8.1+) Returns a list of the user preferred locales in most specific to least specific order, based on the user's language specification.
::msgcat::mcset locale src-string ?translate-string?	(Tcl 8.1+) Sets the translation for src-string to translate-string in the specified locale and the current namespace. If translate-string is not specified, src-string is used for both.
::msgcat::mcunknown locale src-string	(Tcl 8.1+) Used by ::msgcat::mc when src-string is not defined in the current locale. Default action is to return src-string. Can be redefined to do other things.

Tcl 8.5 adds: msgcat::mcpreferences command will be modified to add the empty string as a list element after the elements corresponding to the current locale.

Locale Specification

Defult locale is specified at start-up by checking for the first non-empty value in the ::env(LC_ALL) variable, ::env(LC_MESSAGES) variable , ::env(LANG) variable, and the Windows registry (MS Windows only). Defaults to a locale of "C".

Locale Format	Name	Example
language[_country][_modifier]	The country, language, and system-specific codes.	
language[_country]	The country and language codes.	en_US
language	The language code.	en
{}	(Tcl 8.5+) Root locale	

The country and language codes are specified in standards ISO-639 and ISO-3166

4.4 registry Package

(MS Windows only) The registry package provides a general set of operations for manipulating the Windows registry.

Command	Description
registry broadcast keyName ?**-timeout** milliseconds?	(Tcl 8.4.1+) Sends a broadcast message to the system and running programs to notify them of an update to keyName. Used for environment updates, etc. The timeout specifies how long to wait (default is 3000) for applications to respond to the broadcast message.
registry delete keyName ?valueName?	Deletes valueName from the registry under keyName. Without valueName, keyName and all values under it are deleted. Returns an error if the keyNameor ValueName could not be deleted.
registry get keyName valueName	Returns the data associated with the value valueName under the key keyName. Returns an error if keyName or ValueName doesn't exist. See Supported Data Types below, for the types.
registry keys keyName ?pattern?	Returns a list of names of the subkeys under keyName matching pattern (using Pattern Globbing), if specified, or all subkeys without pattern. Returns an error if keyName doesn't exist.
registry set keyName ?valueName data ?type??	Sets valueName under keyName to data with type type (defult is sz). Creates the key keyName if it doesn't already exist. Without valueName,the key is only created if it doesn't exist. See Supported Data Types below, for the types.
registry type keyName valueName	Returns the type of the value valueName in the key keyName. See Supported Data Types below, for the types.
registry values keyName ?pattern?	Returns a list of names of the values under keyName matching pattern (using Pattern Globbing), if specified, or all values without pattern. Returns an error if keyName doesn't exist.

Key Name Formats

Valid keyName formats (where keypath can be one or more registry key names separated by backslash (\) characters):

\\hostname\rootname\keypath
rootname\keypath
rootname

Root Name Formats:

Valid rootname components:

HKEY_LOCAL_MACHINE
HKEY_USERS
HKEY_CLASSES_ROOT
HKEY_CURRENT_USER
HKEY_CURRENT_CONFIG
HKEY_PERFORMANCE_DATA
HKEY_DYN_DATA

Supported Data Types:

Type	Description	Representation
binary	The registry value contains arbitrary binary data.	binary string
none	The registry value contains arbitrary binary data with no defined type.	binary string
sz	The registry value contains a null-terminated string.	string
expand_sz	The registry value contains a null-terminated string with unexpanded references to env vars in Windows style (eg. "%PATH%")	string
dword	The registry value contains a little-endian 32-bit number.	decimal string
dword_big_endian	The registry value contains a big-endian 32-bit number.	decimal string
link	The registry value contains a symbolic link.	binary string
multi_sz	The registry value contains an array of null-terminated strings.	list of strings
resource_list	The registry value contains a device-driver resource list.	binary string

Unknown types are with the 32-bit integer for that type code returned by the system interfaces and the data is represented as a binary string.

4.5 resource Package

(Mac only)

4.6 tcltest Package

	(Tcl 8.2+)

Appendix A: Command Index

Command	Sect#	Command	Sect#	Command	Sect#	Command	Sect#
after	2.07	fconfigure	2.10	menu	3.16	tcl_endOfWord	2.17
append	2.17	fcopy	2.10	menubutton	3.17	tcl_findLibrary	2.03
array	2.01	file	2.08	message	3.18	tcl_startOfNextWord	2.17
auto_execok	2.03	fileevent	2.08	::msgcat	4.3	tcl_startOfPreviousWord	2.17
auto_import	2.03	flush	2.10	namespace	2.14	tcl_wordBreakAfter	2.17
auto_load	2.03	focus	3.28	open	2.10	tcl_wordBreakBefore	2.17
auto_mkindex	2.03	font	3.9	option	3.19	tcltest	4.6
auto_mkindex_old	2.03	for	2.04	pack	3.11	tell	2.10
auto_qualify	2.14	foreach	2.04	package	2.15	text	3.25
auto_reset	2.03	format	2.17	panedwindow	3.20	time	2.03
bell	3.28	frame	3.10	parray	2.01	tk	3.28
bgerror	2.03	gets	2.10	pid	2.10	tkwait	2.07
binary	2.17	glob	2.10	place	3.11	tk_bisque	3.19
bind	3.1	global	2.16	::pkg	2.15	tk_chooseColor	3.7
bindtags	3.1	grab	3.28	pkg_mkIndex	2.15	tk_chooseDirectory	3.7
break	2.04	grid	3.11	proc	2.16	tk_dialog	3.7
button	3.2	history	2.09	puts	2.10	tk_focusFollowsMouse	3.28
canvas	3.3	::http	4.2	pwd	2.10	tk_focusNext	3.28
case	2.04	if	2.04	radiobutton	3.21	tk_focusPrev	3.28
catch	2.03	image	3.12	raise	3.28	tk_getOpenFile	3.7
cd	2.10	incr	2.18	read	2.10	tk_getSaveFile	3.7
checkbutton	3.4	info	2.11	regexp	2.17	tk_menuSetFocus	3.16
clipboard	3.5	interp	2.12	registry	4.4	tk_messageBox	3.7
clock	2.02	join	2.13	regsub	2.17	tk_optionMenu	3.17
close	2.10	label	3.13	rename	2.03	tk_popup	3.16
concat	2.13	labelframe	3.14	resource	4.5	tk_setPalette	3.19
console	3.6	lappend	2.13	return	2.16	tk_textCopy	3.25
consoleinterp	3.6	lassign	2.13	::safe	2.12	tk_textCut	3.25
continue	2.04	lindex	2.13	scale	3.22	tk_textPaste	3.25
dde	4.1	linsert	2.13	scan	2.17	toplevel	3.26
destroy	3.28	list	2.13	scrollbar	3.23	trace	2.18
dict	2.05	listbox	3.15	seek	2.10	unknown	2.03
encoding	2.06	llength	2.13	selection	3.5	unload	2.03
entry	3.8	load	2.03	send	2.03	unset	2.18
eof	2.10	lower	3.28	set	2.18	update	2.07
error	2.03	lrange	2.13	socket	2.10	uplevel	2.18
eval	2.03	lrepeat	2.13	source	2.03	upvar	2.18
event	3.1	lreplace	2.13	spinbox	3.24	variable	2.14
exec	2.10	lsearch	2.13	split	2.13	vwait	2.07
exit	2.03	lset	2.13	string	2.17	while	2.04
expr	2.03	lsort	2.13	subst	2.03	winfo	3.27
fblocked	2.10	memory	2.11	switch	2.04	wm	3.28