# Tcl Reference Guide

**for Tcl 8.4 and 8.5**

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

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

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

Reference guide reduced to Tcl only and updated to Tcl 8.4 and 8.5 by
Peter Kamphuis **<quickref*(at)*campacasa*(dot)*eu>**

The PDF and LaTeX sources of this reference guide can be found at
**http://linux.campacasa.eu/Tcl_Reference_Guide**

# Contents

# Conventions

| | |
|---|---|
| **fixed** | denotes literal text. |
| *this* | means variable text, i.e. things you must fill in. |
| **word** | is a keyword, i.e. a word with a special meaning. |
| [ ] | denotes an optional part. |
| ... | denotes a repetition. |

Syntax only available with Tcl 8.5 is indicated with "(8.5)". Syntax not available with Tcl 8.5 anymore is indicated with "(8.4)". See *command*(n) manual pages or **http://www.tcl.tk/doc/** for more details on the various Tcl commands.

# 1. Basic Tcl Language Features

| | |
|---|---|
| **;** or *newline* | statement separator |
| \ | statement continuation if last character in line |
| **#** | comments out rest of line (if first non−whitespace character) |
| **var** | simple variable |
| **var(index)** | associative array variable |
| **var(i,j)** | multi-dimensional array variable |
| **$var** | variable substitution (also **${var}xyz**) |
| **[expr 1+2]** | command substitution |
| *char* | backslash substitution (see below) |
| **"hello $a"** | quoting with substitution |
| **{hello $a}** | quoting with no substitution (deferred substitution) |

The only data type in Tcl is a string. However, some commands will interpret arguments as numbers/boolean in which case the formats are

| | |
|---|---|
| Integer: | **123 0xff** (hex) **0377** (octal) |
| Floating Point: | **2.1 3. 6e4 7.91e+16** |
| Boolean: | **true false 0 1 yes no** |

Tcl makes the following backslash substitutions:

| | | | |
|---|---|---|---|
| **a** | audible alert (0x7) | *space* | space |
| **b** | backspace (0x8) | *newline* | space |
| **f** | form feed (0xC) | *ddd* | octal value (*d*=0-7) |
| **n** | newline (0xA) | \x*hh* | hexadecimal value (*h*=0-9, a-f) |
| **r** | carriage return (0xD) | \u*hhhh* | Unicode character (*h*=0-9, a-f) |
| **t** | horizontal tab (0x9) | *c* | replace '*c*' with '*c*' |
| **v** | vertical tab (0xB) | \\ | a backslash |

See **Tcl**(n) manual page for more details.

# 2. Tcl Special Variables

| | |
|---|---|
| **argc** | Number of command line arguments. |
| **argv** | Tcl list of command line arguments. |
| **argv0** | Name of script or command interpreter. |
| **env** | Array where each element name is an enviroment variable. |
| **errorCode** | Error code information from the last Tcl error. |
| **errorInfo** | Describes the stack trace of the last Tcl error. |

| | |
|---|---|
| **tcl_interactive** | Contains 1 when running interactively, 0 otherwise. |
| **tcl_library** | Location of standard Tcl libraries. |
| **tcl_patchLevel** | Current patchlevel of Tcl interpreter. |
| **tcl_pkgPath** | List of directories to search for package loading. |
| **tcl_platform** | Array with elements **byteOrder**, **debug**, **machine**, **os**, **osVersion**, **platform**, **threaded**, **user**, **wordSize**, and **pointerSize**. |
| **tcl_precision** | Number of significant digits to retain when converting floating-point numbers to strings (default 0, meaning using as few as possible). |
| **tcl_traceCompile** | Level of tracing info output during bytecode compilation. |
| **tcl_wordchars** | A regular expression, controlling what are "word" characters. |
| **tcl_nonwordchars** | A regular expression, controlling what are "non-word" characters. |
| **tcl_version** | Current version of Tcl interpreter. |

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

# 3. Operators and Expressions

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

| | |
|---|---|
| **- + ~ !** | unary minus, unary plus, bitwise NOT, logical NOT |
| ****** | exponentiation (8.5) |
| *** / %** | multiply, divide, remainder |
| **+ -** | add, subtract |
| **<< >>** | bitwise shift left, bitwise shift right |
| **< > <= >=** | boolean comparisons |
| **== !=** | boolean equals, not equals |
| **eq ne** | boolean string equals, not equals |
| **in ni** | in list, not in list (8.5) |
| **&** | bitwise AND |
| ^ | bitwise exclusive OR |
| **\|** | bitwise (inclusive) OR |
| **&&** | logical AND |
| **\|\|** | logical OR |
| **x ? y : z** | if **x != 0**, then **y**, else **z** |

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

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

| | | | | | |
|---|---|---|---|---|---|
| **abs** | **ceil** | **exp** | **isqrt** (8.5) | **pow** | **sqrt** |
| **acos** | **cos** | **floor** | **log** | **rand** | **srand** |
| **asin** | **cosh** | **fmod** | **log10** | **round** | **tan** |
| **atan** | **double** | **hypot** | **max** (8.5) | **sin** | **tanh** |
| **atan2** | **entier** (8.5) | **int** | **min** (8.5) | **sinh** | **wide** |
| **bool** (8.5) | | | | | |

See **expr**(n), **mathfunc**(n) (8.5) and **mathop**(n) (8.5) manual pages for more details.

# 4. Regular Expressions

| | |
|---|---|
| *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* |
| *regex*{m} | match *regex* exactly *m* times |
| *regex*{m,} | match *regex* at least *m* times |
| *regex*{m,n} | match *regex* at least *m* and at most *n* times |
| *? +? ?? {m}? {m,}? {m,n}? | "non-greedy" quantifiers, preferring the smallest instead of the largest number of matches |
| . | any single character except newline |
| \d \D | match decimal digit, match anything but decimal digit |
| \s \S | match white space, match anything but white space |
| \w \W | match alphanumeric (letter, digit and underscore), match anything but alphanumeric |
| *m* | (where *m* is a non-zero digit) a back reference |
| *c* | match known backslash substitution character or character *c* even if special |
| ^ | match beginning of string |
| $ | match end of string |
| [abc] | match set of characters |
| [^abc] | match characters not in set |
| [a-z] | match range of characters |
| [^a-z] | match characters not in range |
| ( ) | group expressions, can be used for back references |

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

# 5. Pattern Globbing

| | |
|---|---|
| ? | match any single character |
| * | match zero or more characters |
| [abc] | match set of characters |
| [a-z] | match range of characters |
| *c* | match character *c* |
| {a,b,...} | match any of strings a, b, etc. |
| ~ | home directory (for **glob** command) |
| ~*user* | match *user*'s home directory (for **glob** command) |

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

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

# 6. Control Statements

**break** Abort innermost containing loop command.

**case** Obsolete, see **switch**.

**continue**
Skip to the next iteration of innermost containing loop command.

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

**for** *start test next body*
Looping command where *start*, *next*, and *body* are Tcl command strings and *test* is an expression string to be passed to **expr** command.

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

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

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

**return** [**-code** *code*] [**-errorinfo** *info*] [**-errorcode** *code*] [*string*]
Return immediately from current procedure with *string* as return value.

**switch** [*options*] [--] *string* **{***pattern1 body1* [ *pattern2 body2 ...*] **}**
The *string* argument is matched against each of the *patternN* arguments in order. The *bodyN* of the first match found is evaluated. If no match is found and the last pattern is the keyword **`default`**, its *bodyN* is evaluated. Possible options are **`-exact`**, **`-glob`**, **`-regexp`**, **`-nocase`** (8.5), **`-matchvar`** *varName* (8.5), and **`-indexvar`** *varName* (8.5).

**while** *test body*
Evaluates the Tcl command string *body* as long as expression string *test* evaluates to true.

# 7. File Information

**file atime** *fileName* [*time*]
Time *fileName* was last accessed as seconds since January 1, 1970. Set access time of *fileName* if *time* is specified.

**file attributes** *fileName* [*option* [*value ...* ] ]
Query or set platform-specific attributes of *fileName*. Options are for UNIX: **`-group`**, **`-owner`**, **`-permissions`**; for Windows **`-archive`**, **`-hidden`**, **`-longname`**, **`-readonly`**, **`-shortname`**, **`-system`**; and for MacOS: **`-creator`**, **`-hidden`**, **`-readonly`**, **`-rsrclength`** (8.5), **`-type`**.

**file channels** [*pattern*]
Returns list of all registered open channels, optionally matching *pattern*.

**file copy** [**-force**] [--] *source* [*source ...*] *target*
Makes a copy of *source* under name *target*. If multiple sources are given, *target* must be a directory. Use **`-force`** to overwrite existing files.

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

**file dirname** *pathName*
Returns all directory path components of *pathName*.

**file executable** *pathName*
Returns 1 if *pathName* is executable by user, 0 otherwise.

**file exists** *pathName*
Returns 1 if *pathName* exists (and user can read its directory), 0 otherwise.

**file extension** *pathName*
Returns all characters in *pathName* after and including the last dot.

**file isdirectory** *pathName*
Returns 1 if *pathName* is a directory, 0 otherwise.

**file isfile** *pathName*
Returns 1 if *pathName* is a regular file, 0 otherwise.

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

**file link** [**-symbolic** | **-hard**] *linkName* [*target*]
Create a link *linkName* pointing to *target*. On UNIX the default is a symbolic link.

**file lstat** *pathName varName*
Same as **file stat** except uses the lstat kernel call.

**file mkdir** *dirName* [*dirName* ...]
Creates given directories.

**file mtime** *fileName* [*time*]
Time *fileName* was last modified as seconds since January 1, 1970. Set modified time of *fileName* if *time* is specified.

**file nativename** *fileName*
Returns the platform-specific name of *fileName*.

**file normalize** *pathName*
Returns a unique absolute, resolved and normalized path representation of *pathName*.

**file owned** *pathName*
Returns 1 if *pathName* owned by the current user, 0 otherwise.

**file pathtype** *pathName*
Returns one of **absolute**, **relative**, or **volumerelative**.

**file readable** *pathName*
Returns 1 if *pathName* is readable by current user, 0 otherwise.

**file readlink** *pathName*
Returns value of symbolic link given by *pathName*.

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

**file rootname** *pathName*
Returns all the characters in *pathName* up to but not including last dot.

**file separator** [*pathName*]
Returns character used to separate path segments.

**file size** *fileName*
Returns size of *fileName* in bytes.

**file split** *pathName*
Returns list whose elements are the path components of *pathName*.

**file stat** *pathName varName*
Place results of stat kernel call on *pathName* in variable *varName* as an array with elements **atime**, **ctime**, **dev**, **gid**, **ino**, **mode**, **mtime**, **nlink**, **size**, **type**, and **uid**.

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

**file tail** *pathName*
Return all characters in *pathName* after last directory separator.

**file type** *pathName*
Returns type of *pathName*. Possible values are **file**, **directory**, **characterSpecial**, **blockSpecial**, **fifo**, **link**, or **socket**.

**file volumes**
Returns just "/" on UNIX, list of local drives on Windows, and list of local and network drives on MacOS.

**file writable** *pathName*
Returns 1 if *pathName* is writable by current user, 0 otherwise.

# 8. Tcl Interpreter Information

**info args** *procName*
Returns list describing in order the names of arguments to *procName*.

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

**info cmdcount**
Returns the total number of commands that have been invoked.

**info commands** [*pattern*]
Returns list of all Tcl commands (built-ins and procs), optionally string matching *pattern*.

**info complete** *command*
Returns 1 if *command* is a complete Tcl command, 0 otherwise. 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 otherwise.

**info frame** [*number*]
Provides access to all frames on the stack. (8.5)

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

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

**info hostname**
Returns name of computer on which interpreter was invoked.

**info level** [*number*]
Without *number* returns the stack level of the invoking procedure. Or returns name and arguments of procedure invoked at stack level *number*.

**info library**
Returns name of library directory where standard Tcl scripts are stored.

**info loaded** [*interp*]
Returns list describing packages loaded into *interp*.

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

**info nameofexecutable**
Returns full pathname of binary from which the application was invoked.

**info patchlevel**
Returns current patch level for Tcl.

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

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

**info sharedlibextension**
Returns extension used by platform for shared objects.

**info tclversion**
Returns version number of Tcl in *major.minor* form.

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

# 9. Lists

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

**join** *list* [*joinString*]
Returns string created by joining all elements of *list* with *joinString*.

**lappend** *varName* [*value ...*]
Appends each *value* to the end of the list stored in *varName*.

**lassign** *list varName* [*varName ...*]
Assign list elements to variables *varName*. Too many *varName* will be empty, too few *varName* will return unassigned elements. (8.5)

**lindex** *list* [*index ...*]
Returns value of element at *index* in *list*. Without *index* returns *list*. Multiple *index* allow to select from sublists.

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

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

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

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

**lrepeat** *number element1* [*element2 ...*]
Returns new list consisting of *number* times the sequence of *elementN*. (8.5)

**lreplace** *list first last* [*element ...*]
Returns new list formed by replacing elements *first* through *last* in *list* with given elements.

**lreverse** *list*
Returns new list consisting of elements of *list* in reverse order. (8.5)

**lsearch** [*mode*] *list pattern*
Returns index of first element in *list* that matches *pattern* (-1 for no match). Mode may be **-exact**, **-glob** (default), **-regexp**, or **-sorted**. See **lsearch**(n) manual page for more options.

**lset** *varName* [*index ...*] *newValue*
Replaces an element at *index* in the list stored in *varName* with *newValue*. Multiple *index* allow to assign to sublists. Without *index* replaces the old value of *varName*.

**lsort** [*switches*] *list*
Returns new list formed by sorting *list* according to *switches*. These are

| | |
|---|---|
| **-ascii** | string comparison (default) |
| **-dictionary** | like **-ascii** but ignores case and is number smart. |
| **-integer** | integer comparison |
| **-real** | floating-point comparison |
| **-command** *cmd* | use *cmd* which takes two arguments and returns an integer less than, equal to, or greater than zero |
| **-increasing** | sort in increasing order (default) |
| **-decreasing** | sort in decreasing order |
| **-indices** | return a list of indices into *list* in sorted order instead of the values themselves. (8.5) |
| **-index** *ix* | treats each elements as a sub-list and sorts on the *ix*th element |
| **-nocase** | compare in case-insensitive manner. (8.5) |
| **-unique** | uniquify the sorted list. |

**split** *string* [*splitChars*]
Returns a list formed by splitting *string* at each character in *splitChars* (white-space by default).

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

# 10. Arrays

**array anymore** *arrayName searchId*
Returns 1 if anymore elements are left to be processed in array search *searchId* on *arrayName*, 0 otherwise.

**array donesearch** *arrayName searchId*
Terminates the array search *searchId* on *arrayName*.

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

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

**array names** *arrayName* [*mode*] [*pattern*]
Returns list of all element names in *arrayName*, optionally string matching *pattern*. Mode may be **-exact**, **-glob** (default), or **-regexp**.

**array nextelement** *arrayName searchId*
Returns name of next element in *arrayName* for the search *searchId*.

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

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

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

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

**array unset** *arrayName* [*pattern*]
Unsets all elements in the array *arrayName*, optionally string matching *pattern*.

**parray** *arrayName*
Print to standard output the names and values of all element names in *arrayName*.

# 11. Dictionaries (8.5)

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

**dict append** *dictVariable key* [*string* ...]
Append the given *string*(s) to the value that the given *key* maps to in the dictionary value contained in the given variable, writing the resulting dictionary value back to that variable.

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

**dict exists** *dictValue key* [*key* ...]
Return a boolean value indicating whether the given key exists in the given dictionary value.

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

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

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

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

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

**dict get** *dictValue* [*key* ...]
Get the value of the given *key* in the given dictionary value. Multiple *key*s allow

getting values in nested dictionaries. If no *key*s are given, return a list containing key/value pairs (like **array get**).

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

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

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

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

**dict merge** [*dictValue ...*]
Return a dictionary that contains the contents of each of the *dictValue* arguments.

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

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

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

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

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

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

**dict values** *dictValue* [*globPattern*]
Return a list of all values in the given dictionary value, optionally only those values matching *globPattern* (like **string match**).

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

# 12. Strings and Binary Data

**append** *varName* [*value ...*]
Appends each of the given values to the string stored in *varName*.

**binary format** *formatString* [*arg ...*]
Returns a binary string representation of *arg*s composed according to *formatString*,

a sequence of zero or more field codes each followed by an optional integer count or *****. The possible field codes are:

| | | | |
|---|---|---|---|
| **a** | chars (null padding) | **w** | 64-bit int (little-endian) |
| **A** | chars (space padding) | **W** | 64-bit int (big-endian) |
| **b** | binary (low-to-high) | **m** | as **w\|W**, but native byte order (8.5) |
| **B** | binary (high-to-low) | **f** | float |
| **h** | hex (low-to-high) | **r** | float (little-endian) (8.5) |
| **H** | hex (high-to-low) | **R** | float (big-endian) (8.5) |
| **c** | 8-bit int | **d** | double |
| **s** | 16-bit int (little-endian) | **q** | double (little-endian) (8.5) |
| **S** | 16-bit int (big-endian) | **Q** | double (big-endian) (8.5) |
| **t** | as **s\|S**, but native byte order (8.5) | **x** | nulls |
| **i** | 32-bit int (little-endian) | **X** | backspace |
| **I** | 32-bit int (big-endian) | **@** | absolute position |
| **n** | as **i\|I**, but native byte order (8.5) | | |

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

Extracts values into *varName*'s from binary *string* according to *formatString*. Returns the number of values extracted. Field codes are the same as for **binary format**, except for:

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

**encoding convertfrom** [*encoding*] *data*

Returns *data* converted to Unicode from the specified *encoding* as string.

**encoding convertto** [*encoding*] *string*

Returns *string* converted from Unicode to the specified *encoding* as a sequence of bytes.

**encoding dirs** [*directoryList*]

Sets search path for additional encoding data files, or returns list of directories in search path if *directoryList* is omitted. (8.5)

**encoding names**

Returns list of available encodings.

**encoding system** [*encoding*]

Sets the system encoding, or returns the current system encoding if *encoding* is omitted.

**format** *formatString* [*arg* ...]

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

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

and possible values for *flag* are:

| | | | | | |
|---|---|---|---|---|---|
| **-** | left-justified | **0** | zero padding | **#** | alternate output |
| **+** | always signed | *space* | space padding | | |

**regexp** [*switches*] *exp string* [*matchVar*] [*subMatchVar* ...]

Returns 1 if the regular expression *exp* matches part or all of *string*, 0 otherwise. If specified, *matchVar* will be set to all the characters in the match and the following *subMatchVar*'s will be set to matched parenthesized subexpressions.

The **-nocase** switch can be specified to ignore case in matching. The **-indices** switch can be specified so that *subMatchVar* will be set to a list

containing the start and ending indices in *string* of the corresponding match. See **regexp**(n) manual page for more switches.

**regsub** [*switches*] *exp string subSpec* [*varName*]
Replaces the first portion of *string* that matches the regular expression *exp* with *subSpec*. If *varName* is specified, it will contain the result and the number of replacements made is returned. If *varName* is not specified, the result is returned.

The **-nocase** switch can be specified to ignore case in matching. The **-all** switch will cause all matches to be substituted for. See **regsub**(n) manual page for more switches.

**scan** *string formatString* [*varName* ...]
Extracts values into given variables using ANSI C **sscanf** behavior. Returns the number of values extracted. If no *varName* is specified, returns a list with the extracted data.

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

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

**string bytelength** *string*
Returns the number of bytes used to represent *string* in memory.

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

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

**string first** *string1 string2* [*startIndex*]
Returns index in *string2* of first occurance of *string1* (-1 if not found). Optionally at or after index *startIndex*.

**string index** *string charIndex*
Returns the *charIndex*'th character in *string*.

**string is** *class* [**-strict**] [**-failindex** *varName*] *string*
Returns 1 if *string* is a valid member of the specified character *class*, or 0 when not. See **string**(n) manual page for more information.

**string last** *string1 string2* [*lastIndex*]
Returns index in *string2* of last occurance of *string1* (-1 if not found). Optionally at or before index *lastIndex*.

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

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

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

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

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

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

**string reverse** *string*
Returns *string* with its characters in reverse order. (8.5)

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

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

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

**string trim** *string* [*chars*]
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 charIndex*
Returns index of character just after last one in word at *charIndex* in *string*.

**string wordstart** *string charIndex*
Returns index of first character of word at *charIndex* in *string*.

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

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

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

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

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

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

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

# 13. System Interaction

**cd** [*dirName*]
Change working directory to *dirName*.

**clock add** *timeVal* [*count unit*...] [*option value*]
Adds an offset to a time *timeVal* expressed as an integer number of seconds. As *unit* one of the words **seconds**, **minutes**, **hours**, **days**, **weeks**, **months**, or **years**, or any unique prefix of such word can be used. (8.5)

**clock clicks** [*resolution*]
Returns hi-res system-dependent integer time value. If *resolution* is **-milliseconds**, the value is guaranteed to be of millisecond granularity (obsolete in 8.5, use **clock milliseconds** instead). A *resolution* **-microseconds** (8.5) is synonymous with **clock microseconds** and is obsolete. Use **clock microseconds** instead.

**clock format** *timeVal* [*option value*]
Convert integer *timeVal*, as seconds since 1 January 1970, 00:00 UTC, to human-readable format. The option **-format** *format* recognizes the following placeholders (names as for the given locale):

| | | | |
|---|---|---|---|
| %a | weekday (abbr) | %n | newline (8.4) |
| %A | weekday (full) | %N | month (1 – 12) (8.5) |
| %b | month (abbr) | %Od,%Oe,%OH,%OI,%Ok,%Ol,%Om, | |
| %B | month (full) | %OM,%OS,%Ou,%Ow,%Oy | |
| %c | locale date & time | | locale as without "O" (8.5) |
| %C | century | %p | locale AM/PM |
| %d | day (01 – 31) | %P | locale am/pm (8.5) |
| %D | %m/%d/%Y | %Q | internal use (8.5) |
| %e | day (1 – 31) | %r | locale 12hr time |
| %Ec | locale date & time (8.5) | %R | locale 24hr time |
| %EC | locale era (8.5) | %s | *timeVal* |
| %EE | B.C.E or C.E. (8.5) | %S | seconds (00 – 59) |
| %Ex | locale alternate date (8.5) | %t | TAB |
| %EX | locale alternate time (8.5) | %T | %H:%M:%S |
| %Ey | locale alternate year (00 – 99) (8.5) | %u | weekday (1 – 7) |
| %EY | locale alternate year (full) (8.5) | %U | week (00 – 53) |
| %g | ISO8601 year (00 – 99) | %V | ISO8601 week (01 – 53) |
| %G | ISO8601 year (full) | %w | weekday (0 – 6) |
| %h | month (abbr) | %W | week (00 – 53) |
| %H | hour (00 – 23) | %x | locale date |
| %I | hour (01 – 12) | %X | locale time |
| %j | day (001 – 366) | %y | year (00 – 99) |
| %J | Julian day (8.5) | %Y | year (full) |
| %k | hour (0 – 23) | %z | time zone (±hhmm) (8.5) |
| %l | hour (1 – 12) | %Z | time zone name |
| %m | month (01 – 12) | %% | % |
| %M | minute (00 – 59) | %+ | "%a %b %e %T %Z %Y" (8.5) |

The default format is "%a %b %d %T %Z %Y" ("%a %b %d %T %z %Y" in 8.5).

**clock microseconds**
Returns the current time as an integer number of microseconds. (8.5)

**clock milliseconds**
Returns the current time as an integer number of milliseconds. (8.5)

**clock scan** *dateString* [**-base** *timeVal*] [*option value*]

Convert *dateString* to an integer clock value. If *dateString* contains a 24 hour time only, the date given by *timeVal* is used.

**clock seconds**

Returns the current time as an integer number of seconds.

**Note:** for clock arithmetic, formatting, and scanning **-gmt** *boolean* specifies that a time should be processed in UTC, or defaults to the local time zone (obsolete in 8.5, **-timezone** should be used instead). **-locale** *localeName* (8.5) specifies that locale-dependent processing is to be done. **-timezone** *zoneName* (8.5) specifies that processing is to be done for the given time zone.

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

Execute subprocess using each *arg* as word for a shell pipeline and return results written to standard out, optionally retaining the final newline char. With **-ignorestderr** (8.5) output to standard error will not be treated as error. The following constructs can be used to control I/O flow.

| | |
|---|---|
| **\|** | pipe (stdout) |
| **\|&** | pipe (stdout and stderr) |
| **<***fileName* | stdin from file |
| **<@** *fileId* | stdin from open file |
| **<<***value* | pass value to stdin |
| **>***fileName* | stdout to file |
| 2**>***fileName* | stderr to file |
| **>&** *fileName* | stdout and stderr to file |
| **>>***fileName* | append stdout to file |
| 2**>>***fileName* | append stderr to file |
| **>>&** *fileName* | append stdout and stderr to file |
| **>@** *fileId* | stdout to open file |
| 2**>@** *fileId* | stderr to open file |
| 2**>**@1 | redirect stderr to stdout (8.5) |
| **>&@** *fileId* | stdout and stderr to open file |
| **&** | run in background |

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

Returns list of all files in current directory that match any of the given csh-style glob patterns. The following *switches* are supported.

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

**pid** [*fileId*]
Return process id of process pipeline *fileId* if given, otherwise return process id of interpreter process.

**pwd** Returns the absolute path name of the current working directory.

# 14. File Input/Output

**close** *channelId*
Close the open file channel *channelId*.

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

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

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

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

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

**-buffersize** *byteSize*
Size of buffer. Minimum value is 10, maximum value is 1 million.

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

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

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

For socket channels (read-only settings):

**-error**
Get the current error status of the socket.

**-sockname**
Returns three element list with address, host name and port number.

**-peername**
For client and accepted sockets, three element list of peer socket.

See **open**(n) manual page for details on **fconfigure** options for serial device channels.

**fcopy** *inId outId* [**-size** *size*] [**-command** *callback*]
Copy data from *inId* to *outId* until eof or *size* bytes have been transferred. If **-command** is given, copy occurs in background and runs *callback* when finished, appending number of bytes copied and possible error message as arguments.

**fileevent** *channelId* **readable | writable** [*script*]
Evaluate *script* when channel *channelId* becomes readable/writable.

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

**gets** *channelId* [*varName*]
Read next line from channel *channelId*, discarding newline character. If *varName* is not given, returns the characters read. If *varName* is given, places the characters read in it and returns the number of characters read.

**open** *fileName* [*access* [*perms*]]
Opens *filename* and returns its channel id. If a new file is created, its permission are set to the conjuction of *perms* (defaulting to **0666**) and the process umask. The *access* may be

- **r** Read only. File must exist. Default if *access* is not given.
- **r+** Read and write. File must exist.
- **w** Write only. Truncate if exists.
- **w+** Read and write. Truncate if exists.
- **a** Write only. Create new empty file if not existing yet. Access position at end.
- **a+** Read and write. Create new empty file if not existing yet. Access position at end.

**puts** [**-nonewline**] [*channelId*] *string*
Write string to *channelId* (default **stdout**), optionally omitting newline char.

**read** [**-nonewline**] *channelId*
Read all remaining bytes from *channelId*, optionally discarding last character if it is a newline.

**read** *channelId numBytes*
Read *numBytes* bytes from *channelId*.

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

**socket** [*option ...*] *host port*
Open a client-side TCP socket to server *host* on *port* and returns its channel identifier. Options are:

- **-myaddr** *addr* Set network address of client (if multiple available).
- **-myport** *port* Set connection port of client (if different from server).
- **-async** Make connection asynchronous.

**socket -server** *command* [**-myaddr** *addr*] *port*
Open server TCP socket on *port* invoking *command* once connected with three arguments: the channel, the address, and the port number.

**tell** *channelId*
Return current access position in *channelId*.

# 15. Channels (8.5)

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

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

**chan close** *channelId*
Close and destroy channel *channelId*.

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

- **-blocking** *boolean* Whether I/O can block process.
- **-buffering full|line|none** How to buffer output.

**-buffersize** *byteSize*
Size of buffer. Maximum value is 1 million.

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

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

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

For socket channels (read-only settings):

**-error**
Get the current error status of the socket.

**-sockname**
Returns three element list with address, host name and port number.

**-peername**
For client and accepted sockets, three element list of peer socket.

See **open**(n) manual page for details on **chan configure** options for serial device channels.

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

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

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

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

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

**chan gets** *channelId* [*varName*]
Read next line from channel *channelId*, discarding newline character. If *varName* is not given, returns the characters read. If *varName* is given, places the characters read in it and returns the number of characters read.

**chan names** [*pattern*]
Returns a list of all channel names, optionally only those matching *pattern* (like **string match**).

**chan pending** *mode channelId*
Depending on whether *mode* is **input** or **output**, returns the number of bytes of input or output currently buffered internally for channel channelId.

**chan postevent** *channelId eventSpec*
This subcommand is used by command handlers specified with **chan create**. It

notifies the channel represented by the handle *channelId* that the event(s) listed in the *eventSpec* have occurred. The argument has to be a list containing any of the strings **read** and **write**. See **refchan**(n) manual page for more details.

**chan puts** [**-nonewline**] [*channelId*] *string*
Write string to channel *channelId* (default **stdout**), optionally omitting newline char.

**chan read** *channelId* [*numChars*]
Read *numBytes* characters from channel *channelId*.

**chan read** [**-nonewline**] *channelId*
Read all remaining bytes from channel *channelId*, optionally discarding last character if it is a newline.

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

**chan tell** *channelId*
Return current access position in channel *channelId*.

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

# 16. Multiple Interpreters

**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 *arg*s in the interpreter *targetPath*.

**interp aliases** [*path*]
Returns list of all aliases defined in interpreter *path*.

**interp bgerror** *path* [*cmdPrefix*]
Get or set the current background error handler for interpreter *path*. (8.5)

**interp create** [**-safe**] [--] [*path*]
Creates a slave interpreter (optionally safe) named *path*.

**interp debug** *path* [**-frame** [*bool*]]
Controls whether to capture frame-level stack information in slave interpreter *path*. (8.5)

**interp delete** *path* [*path ...*]
Deletes the interpreter(s) *path* and all its slave interpreters.

**interp eval** *path arg* [*arg ...*]
Evalutes concatenation of *arg*s as command in interpreter *path*.

**interp exists** *path*
Returns 1 if interpreter *path* exists, 0 otherwise.

**interp expose** *path hiddenCmd* [*exposedCmd*]
Make *hiddenCmd* in interpreter *path* exposed (optionally as *exposedCmd*).

**interp hide** *path exposedCmd* [*hiddenCmd*]
Make *exposedCmd* in interpreter *path* hidden (optionally as *hiddenCmd*).

**interp hidden** *path*
Returns list of hidden commands in interpreter *path*.

**interp invokehidden** *path* [*option ...*] *hiddenCmd* [*arg ...*]
Invokes *hiddenCmd* with specified *arg*s in interpreter *path*. Options are:

| | |
|---|---|
| **-global** | Invoke at global level. |
| **-namespace** *name* | Invoke in namespace *name*. (8.5) |
| **--** | Allows *hiddenCmd* to start with **-**. (8.5) |

**interp issafe** [*path*]
Returns 1 if interpreter *path* is safe, 0 otherwise.

**interp limit** *path limitType* [*-option*] [*value ...*]
Sets up, manipulates and queries the configuration of resource limit *limitType* for interpreter *path*. (8.5)

**interp marktrusted** [*path*]
Marks interpreter *path* as trusted.

**interp recursionlimit** *path* [*newLimit*]
Returns the maximum allowable nesting depth for interpreter *path*, optionally setting a new limit.

**interp share** *srcPath channelId destPath*
Arranges for 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*.

**interp target** *path alias*
Returns Tcl list describing target interpreter of *alias* in interpreter *path*.

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

For each slave interpreter created, a new Tcl command is created by the same name in its master. This command has the **alias**, **aliases**, **bgerror** (8.5), **eval**, **expose**, **hide**, **hidden**, **invokehidden**, **issafe**, **limit** (8.5), **marktrusted**, and **recursionlimit** subcommands like **interp**, but without the *srcPath* and *path* arguments (they default to the slave itself) and without the *targetPath* argument (it defaults to the slave's master).
A safe interpreter is created with the following commands exposed:

| | | | | |
|---|---|---|---|---|
| **after** | **error** | **info** | **lsort** | **split** |
| **append** | **eval** | **interp** | **namespace** | **string** |
| **apply (8.5)** | **expr** | **join** | **package** | **subst** |
| **array** | **fblocked** | **lappend** | **pid** | **switch** |
| **binary** | **fcopy** | **lassign (8.5)** | **proc** | **tell** |
| **break** | **fileevent** | **lindex** | **puts** | **time** |
| **catch** | **flush** | **linsert** | **read** | **trace** |
| **chan (8.5)** | **for** | **list** | **regexp** | **unset** |
| **clock** | **foreach** | **llength** | **regsub** | **update** |
| **close** | **format** | **lrange** | **rename** | **uplevel** |
| **concat** | **gets** | **lrepeat (8.5)** | **return** | **upvar** |
| **continue** | **global** | **lreplace** | **scan** | **variable** |
| **dict (8.5)** | **if** | **lsearch** | **seek** | **vwait** |
| **eof** | **incr** | **lset (8.5)** | **set** | **while** |

A safe interpreter is created with the following commands hidden:

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

# 17. Packages

**package forget** *package*
Remove all info about *package* from interpreter.

**package ifneeded** *package version* [*script*]
Tells interpreter that if version *version* of *package*, evaluating *script* will provide it.

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

**package prefer** [**latest** | **stable**]
Return or set the **package require** selection logic mode. (8.5)

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

**package provide** *package* [*version*]
Tells interpreter that *package version* is now provided. Without *version*, the currently provided version of *package* is returned.

**package require** *package* [*requirement*...]
Tells interpreter that a suitable *package* must be provided. A suitable package must satisfy at least one of the *requirement*s as per **package vsatisfies** rules. The version number of the package loaded is returned. (8.5)

**package require** [**-exact**] *package* [*version*]
Tells interpreter that *package* (with the exact *version*) must be provided.

**package unknown** [*command*]
Specifies a last resort Tcl command to provide a package, appending the desired package and version or requirements.

**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* with an **ifneeded** script.

**package vsatisfies** *version requirement*...
Returns 1 if *version* satisfies at least one of *requirement*s, 0 otherwise. Requirements are in the form (where **`min`** and **`max`** are valid version numbers, 8.5):

| | |
|---|---|
| **`min`** | Min-bounded. |
| **`min-`** | Min-unbound. |
| **`min-max`** | Bounded. |

**package vsatisfies** *version1 version2*
Returns 1 if *version2* scripts will work unchanged under *version1*, 0 otherwise.

# 18. Namespaces

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

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

**namespace current**
Returns fully-qualified name of current namespace.

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

**namespace ensemble** *subcommand* [*arg ...*]
Creates and manipulates a command that is formed out of an ensemble of subcommands. (8.5)

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

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

**namespace export [-clear]** [*pattern ...*]
Adds to export list of current namespace all commands that match given *pattern*'s. If **-clear** is given, the export list is first emptied.

**namespace forget** [*namespace*::*pattern ...*]
Removes from current namespace any previously imported commands from *namespace* that match *pattern*.

**namespace import [-force]** [*namespace*::*pattern ...*]
Imports into current namespace commands matching *pattern* 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 of *arg*'s to *listArg*.

**namespace origin** *command*
Returns fully-qualified name of imported *command*.

**namespace parent** [*namespace*]
Returns fully-qualified name of parent namespace of *namespace*.

**namespace path** [*namespaceList*]
Returns or sets the command resolution path of the current namespace. (8.5)

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

**namespace tail** *string*
Returns the simple name (strips namespace qualifiers) in *string*.

**namespace upvar** *namespace otherVar myVar* [*otherVar myVar ...*]
Arrange for one or more local variables in the current procedure to refer to variables in *namespace*. (8.5)

**namespace unknown** [*script*]
Sets or returns the unknown command handler for the current namespace. (8.5)

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

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

# 19. Other Tcl Commands

**after** *ms* [*arg1 arg2 arg3 ...*]
Arrange for command (concat of *args*) to be run after *ms* milliseconds have passed.

With no *args*, program will sleep for *ms* milliseconds. Returns the id of the event handler created.

**after cancel** *id* | *arg1 arg2 ...*
Cancel previous **after** command either by command or the id returned.

**after idle** [*arg1 arg2 arg3 ...*]
Arrange for command (concat of *args*) to be run later when there are no events to process in the (Tk) event loop. Returns the id of the event handler created.

**after info** [*id*]
Returns information on event handler *id*. With no *id*, returns a list of all existing event handler ids.

**apply** *function* [*arg ...*]
Apply *function* to the given arguments and return the result. (8.5)

**auto_execok** *command*
Returns a list of arguments to be passed to **exec** if an executable file or shell builtin by the name *command* exists in user's PATH, empty string otherwise.

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

**auto_load** *command*
Attempts to load definition for *command* by searching **`$auto_path`** and **`$env(TCLLIBPATH)`** for a tclIndex file which will inform the interpreter where it can find *command*'s definition.

**auto_mkindex** *directory pattern* [*pattern ...*]
Generate a tclIndex file from all files in *directory* that match glob patterns.

**auto_qualify** *command namespace*
Compute a list of fully qualified names for *command*.

**auto_reset**
Destroys cached information used by **auto_execok** and **auto_load**.

**bgerror** *message*
User defined handler for background Tcl errors. (Preferrably use **interp bgerror** in 8.5.)

**catch** *script* [*resultVarName*] [*optionsVarName*]
Evaluate *script* and optionally store results into *resultVarName*. Optionally store a directory of return options into *optionsVarName* (8.5). If there is an error, a non-zero error code is returned and an error message stored in *resultVarName*.

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

**error** *message* [*info*] [*code*]
Interrupt command interpretation with an error described in *message*. Global variables **`errorInfo`** and **`errorCode`** will be set to *info* and *code*.

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

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

**global** *varName* [*varName ...*]
Declares given *varName*'s as global variables.

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

**incr** *varName* [*increment*]
: Increment the integer value stored in *varName* by *increment* (default 1). If *varName* is unset, set it to *increment* or to 1 by default (8.5).

**load** *file* [*pkgName* [*interp*]]
: Load binary code for *pkgName* from *file* (dynamic lib) into *interp*.

**memory** *command* [*args*]
: Control Tcl memory debugging capabilities. See **memory**(n) manual page for more details.

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

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

**pkg_mkIndex** [**-direct**] [**-lazy**] [**-load** *pkgPat*] [**-verbose**] *dir* [*pattern* ...]
: Create index files that allow packages to be loaded automatically when **package require** commands are executed.

**platform::***command* [*arg*]
: The **platform** package provides several utility commands useful for the identification of the architecture of a machine running Tcl. See **platform**(n) manual page for more details. (8.5)

**platform::shell::***command shell*
: The **platform:shell** package provides several utility commands useful for the identification of the architecture of a specific Tcl shell. See **platform::shell**(n) manual page for more details. (8.5)

**proc** *name args body*
: Create a new Tcl procedure (or replace existing one) called *name* where *args* is a list of arguments and *body* Tcl commands to evaluate when invoked. If the last argument has the name **`args`**, then this will be a list containing the values of any remaining arguments when invoked.

**registry** *command args*
: Manipulate the Microsoft Windows registry. See **registry**(n) manual page for more details.

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

**set** *varName* [*value*]
: Store *value* in *varName* if given. Returns the current value of *varName*.

**source** [**-encoding** *encoding*] *fileName*
: Read file *fileName* and evaluate its contents as a Tcl *script*. The encoding of *fileName* can be specified (8.5).

**::tcl::tm::***command args*
: Facilities for locating and loading of Tcl Modules. See **tm**(n) manual page for more details. (8.5)

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

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

**time** *script* [*count*]
: Call interpreter *count* (default 1) times to evaluate *script*. Returns string of the form "`503 microseconds per iteration`".

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

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

**unload** [**-nocomplain**] [**-keeplibrary**] [--] *file* [*pkgName* [*interp*]]
: Try to unload shared libraries previously loaded with **load**. (8.5)

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

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

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

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

**vwait** *varName*
: Enter Tcl event loop until global variable *varName* is modified.

# Command Index

# Notes