Tcl Reference Guide

for Tcl 8.0

Tk/Tcl program designed and created by
John Ousterhout **<ouster@scriptics.com>**

Reference guide contents written by
Paul Raines **<raines@slac.stanford.edu>**
Jeff Tranter **<tranter@pobox.com>**

Reference guide format designed and created by
Johan Vromans **<jvromans@squirrel.nl>**

Reference guide reduced to Tcl only by
Peter Kamphuis **<peter.kamphuis@infineon.com>**

Contents

Rev. 8.0.3

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.

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*dd*	hexadecimal value (*d*=0-9, a-f)
r	carriage return (0xD)	*c*	replace '*c*' with '*c*'
t	horizontal tab (0x9)	\\	a backslash
v	vertical tab (0xB)		

2. Tcl Special Variables

argc	Number of command line arguments.
argv	Tcl list of command line arguments.
arg0	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_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**, **osVersion**, **machine**, **platform**, and **os**.

tcl_precision	Number of significant digits to retain when converting floating-point numbers to strings (default 12).
tcl_traceCompile	Level of tracing info output during bytecode compilation.
tcl_traceExec	Level of tracing info output during bytecode execution.
tcl_version	Current version of Tcl interpreter.

3. Operators and Expressions

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

- ~ !	unary minus, bitwise NOT, logical NOT
*** / %**	multiply, divide, remainder
+ -	add, subtract
<< >>	bitwise shift left, bitwise shift right
< > <= >=	boolean comparisons
== !=	boolean equals, not equals
&	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**	**floor**	**log10**	**sinh**
acos	**cos**	**fmod**	**pow**	**sqrt**
asin	**cosh**	**hypot**	**rand**	**srand**
atan	**double**	**int**	**round**	**tan**
atan2	**exp**	**log**	**sin**	**tanh**

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*
.	any single character except newline
^	match beginning of string
$	match end of string
c	match character *c* even if special
[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

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.

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*.

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 *pattern* 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`**, and **`-regexp`**.

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 *fileName* was last accessed as seconds since Jan. 1, 1970.

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`**, **`-type`**.

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] [- -] *fileName* [*fileName* ...]
Removes given files. Use **-force** to remove non-empty directories.

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

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

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

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 otherwise.

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

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

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

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

file mtime *fileName*
Time *fileName* was last modified as seconds since Jan. 1, 1970.

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

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

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

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

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

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 *fileName*
Returns all the characters in *fileName* up to but not including last dot.

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 variable *varName* as an array with elements **`atime`**, **`ctime`**, **`dev`**, **`gid`**, **`ino`**, **`mode`**, **`mtime`**, **`nlink`**, **`size`**, **`type`**, and **`uid`**.

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 volume
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 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 Tcl commands (built-ins and procs) matching glob *pattern*. If no pattern is given, returns all commands in current namespace.

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 globals [*pattern*]
Returns list of global variables matching glob *pattern* (default *****).

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

info level
Returns the stack level of the invoking procedure.

info level *number*
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 matching glob *pattern* (default *****).

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 matching glob *pattern* (default *****).

info script
Returns name of Tcl script currently being evaluated.

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 matching glob *pattern* (default *****).

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

lindex *list index*
Returns value of element at *index* in *list*.

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

list [*arg 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.

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

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

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.
-index *ndx*	treats each elements as a sub-list and sorts on the *ndx*th element
-integer	integer comparison

-real	floating-point comparison
-increasing	sort in increasing order (default)
-decreasing	sort in decreasing order
-command *cmd*	Use *cmd* which takes two arguments and returns an integer less than, equal to, or greater than zero

split *string* [*splitChars*]
Returns a list formed by splitting *string* at each character in *splitChars*.

Note: list indices start at 0 and the word **end** may be used to reference the last element in the list.

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*
Returns a list where each odd element is an element name and the following even element its corresponding value.

array names *arrayName* [*pattern*]
Returns list of all element names in *arrayName* that match glob *pattern*.

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*.

parray *arrayName* [*pattern*]
Print to standard output the names and values of all element names in *arrayName* that match glob *pattern*.

11. Strings and Binary Data

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

binary format *formatString* [*arg 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. The possible field codes are:

a	chars (null padding)	**c**	8-bit int	**f**	float
A	chars (space padding)	**s**	16-bit int (little)	**d**	double
b	binary (low-to-high)	**S**	16-bit int (big)	**x**	nulls
B	binary (high-to-low)	**i**	32-bit int (little)	**X**	backspace
h	hex (low-to-high)	**I**	32-bit int (big)	**@**	absolute position
H	hex (high-to-low)				

binary scan *string formatString varName* [*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

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

Returns a formated 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 *matchVar* and *subMatchVar* will be set to the start and ending indices in *string* of their corresponding match.

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

Replaces the first portion of *string* that matches the regular expression *exp* with *subSpec* and places results in *varName*. Returns count of number of replacements made. The **-nocase** switch can be specified to ignore case in matching. The **-all** switch will cause all matches to be substituted for.

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

Extracts values into given variables using ANSI C **sscanf** behavior. Returns the number of values extracted. Placeholders have the form **%**[*****][*width*]*char* where ***** is for discard, *width* is an integer and possible values for *char* are:

d	decimal	**e**	float	**s**	string (non-whitespace)
o	octal	**f**	float	**[***chars***]**	chars in given range
x	hex	**g**	float	**[^***chars***]**	chars not in given range
c	char to int				

string compare *string1 string2*

Returns -1, 0, or 1, depending on whether *string1* is lexicographically less than, equal to, or greater than *string2*.

string first *string1 string2*

Return index in *string2* of first occurance of *string1* (-1 if not found).

string index *string charIndex*

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

string last *string1 string2*
Return index in *string2* of last occurance of *string1* (-1 if not found).

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

string match *pattern string*
Returns 1 if glob *pattern* matches *string*, 0 otherwise.

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

string tolower *string*
Returns new string formed by converting all chars in *string* to lower case.

string toupper *string*
Returns new string formed by converting all chars in *string* to upper case.

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

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 of character just after last one in word at *index* in *string*.

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

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

12. System Interaction

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

clock clicks
Returns hi-res system-dependent integer time value.

clock format *clockVal* [**-format** *string*] [**-gmt** *boolean*]
Convert integer *clockVal* to human-readable format defined by *string* which recognizes (at least) the following placeholders:

%%	%	%H	hour (00 – 23)	%U	week (01 – 52)
%a	weekday (abbr)	%h	hour (00 – 12)	%w	weekday (0 – 6)
%A	weekday (full)	%j	day (001 – 366)	%x	locale date
%b	month (abbr)	%m	month (01 – 12)	%X	locale time
%B	month (full)	%M	minute (00 – 59)	%y	year (00 – 99)
%c	locale date & time	%p	AM/PM	%Y	year (full)
%d	day (01 – 31)	%S	seconds (00 – 59)	%Z	time zone

The default format is "%a %b %d %H:%M:%S %Z %Y".

clock scan *dateString* [**-base** *clockVal*] [**-gmt** *boolean*]
Convert *dateString* to an integer clock value. If *dateString* contains a 24 hour time only, the date given by *clockVal* is used.

clock seconds
Return current date and time as system-dependent integer value.

exec [**-keepnew**] *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. 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*	stdout and stderr to file
>@ *fileId*	stdout to open file
2**>@** *fileId*	stderr to open file
>&@ *fileId*	stdout and stderr to open file
&	run in background

glob [**-nocomplain**] *pattern* [*pattern* ...]
Returns list of all files in current directory that match any of the given csh-style glob patterns, optionally suppressing error on no match.

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

pwd Returns the current working directory.

13. File Input/Output

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

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

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

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

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

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

-buffersize *byteSize* Size of buffer.

-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):

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

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

For serial device channels:

-mode *baud,parity,data,stop*
Set baud rate, parity, data bits, and stop bits of channel.

fcopy *inId outId* [**-size** *size*] [**-command** *callback*]
Transfer data to *outId* from *inId* 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 *fileId* **readable|writable** [*script*]
Evaluate *script* when channel *fileId* becomes readable/writable.

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

gets *fileId* [*varName*]
Read next line from channel *fileId*, discarding newline character. Places characters of line in *varName* if given, otherwise returns them.

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* and the process umask. The *access* may be

r Read only. File must exist.

r+ Read and write. File must exist.

w Write only. Truncate if exists.

w+ Read and write. Truncate if exists.

a Write only. File must exist. Access position at end.

a+ Read and write. Access position at end.

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

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

read *fileId numBytes*
Read *numBytes* bytes from *fileId*.

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

socket [*option ...*] *host port*
Open a client-side TCP socket to server *host* on *port*. 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 *fileId*
Return current access position in *fileId*.

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

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 interp *path* exposed (optionally as *exposedCmd*).

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

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

interp invokehidden *path* [**-global**] *hiddenCmd* [*arg* ...]
Invokes *hiddenCmd* with specified *arg*s in interp *path* (at the global level if **-global** is given).

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

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

interp share *srcPath fileId destPath*
Arranges for I/O channel *fileId* 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 fileId destPath*
Moves I/O channel *fileId* 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**, **eval**, **expose**, **hide**, **hidden**, **invokehidden**, **issafe**, and **marktrusted** 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	**eval**	**incr**	**lsearch**	**split**
append	**expr**	**info**	**lsort**	**string**
array	**fblocked**	**interp**	**package**	**subst**

break	**fileevent**	**join**	**pid**	**switch**
case	**flush**	**lappend**	**proc**	**tell**
catch	**for**	**lindex**	**puts**	**trace**
clock	**foreach**	**linsert**	**read**	**unset**
close	**format**	**list**	**rename**	**update**
concat	**gets**	**llength**	**return**	**uplevel**
continue	**global**	**lower**	**scan**	**upvar**
eof	**history**	**lrange**	**seek**	**vwait**
error	**if**	**lreplace**	**set**	**while**

A safe interpreter is created with the following commands hidden:

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

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

package require [`-exact`] *package* [*version*]
Tells interpreter that *package* must be provided. Only packages with versions equal to or later than *version* (if provided) are acceptable. If `-exact` is specified, the exact version specified must be provided.

package unknown [*command*]
Specifies a last resort Tcl command to provide a package which have append as its final two arguments the desired package and version.

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

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

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

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

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 outsize a **namespace eval**, a local variable is created linked to the given namespace variable.

17. 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 Tk is idle. 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.

auto_execok *execFile*

Returns full pathname if an executable file by the name *execFile* exists in user's PATH, empty string otherwise.

auto_load *command*

Attempts to load definition for *cmd* 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_reset

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

bgerror *message*

User defined handler for background Tcl errors. Default exists for Tk.

catch *script* [*varName*]

Evaluate *script* and store results into *varName*. If there is an error, a non-zero error code is returned and an error message stored in *varName*.

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* for more info.

global *varName* [*varName* ...]

Declares given *varName*'s as global variables.

history add *command* [**`exec`**]

Adds *command* to history list, optionally executing it.

history change *newValue* [*event*]

Replaces value of *event* (default current) in history with *newValue*.

history clear

Erase the history list and reset event numbers.

history event [*event*]

Returns value of *event* (default -1) in history.

history info [*count*]

Returns event number and contents of the last *count* events.

history keep [*count*]

Set number of events to retain in history to *count*.

history nextid

Returns number for next event to be recorded in history.

history redo [*event*]

Re-evaluates *event* (default -1).

incr *varName* [*increment*]

Increment the integer value stored in *varName* by *increment* (default 1).

load *file* [*pkgName* [*interp*]]

Load binary code for *pkgName* from *file* (dynamic lib) into *interp*.

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 commmands to evaluate when invoked.

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 *fileName*
: Read file *fileName* and evaluate its contents as a Tcl *script*.

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

trace variable *varName ops command*
: Arrange for *command* to be evaluated whenever *varName* is accessed in one of the ways specified with *ops*. Possbile values are **`r`** for read, **`w`** for written, **`u`** for unset, and any combination of the three.

trace vdelete *varName ops command*
: Remove any previous trace specified with the given arguments.

trace vinfo *varName*
: Returns list describing each trace on *varName*.

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

unset *varName* [*varName* ...]
: Removes the given variables and arrays from scope.

update [**`idletasks`**]
: Handle pending 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 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

