NAME

argc, argv, argv0, auto_path, env, errorCode, errorInfo, tcl_interactive, tcl_library, tcl_nonwordchars, tcl_patchLevel, tcl_pkgPath, tcl_platform, tcl_precision, tcl_rcFileName, tcl_traceCompile, tcl_traceExec, tcl_wordchars, tcl_version − Variables used by Tcl

DESCRIPTION

The following global variables are created and managed automatically by the Tcl library. Except where noted below, these variables should normally be treated as read-only by application-specific code and by users.

auto_path

If set, then it must contain a valid Tcl list giving directories to search during auto-load operations (including for package index files when using the default **package unknown** handler). This variable is initialized during startup to contain, in order: the directories listed in the **TCLLIBPATH** environment variable, the directory named by the **tcl_library** global variable, the parent directory of **tcl_library**, **[file dirname [file dirname [info nameofexecutable]]]/lib**, the directories listed in the **tcl_pkgPath** variable. Additional locations to look for files and package indices should normally be added to this variable using **lappend**.

Additional variables relating to package management exist. More details are listed in the **VARIABLES** section of the **library** manual page.

env This variable is maintained by Tcl as an array whose elements are the environment variables for the process. Reading an element will return the value of the corresponding environment variable. Setting an element of the array will modify the corresponding environment variable or create a new one if it does not already exist. Unsetting an element of**en v** will remove the corresponding environment variable. Changes to the**en v** array will affect the environment passed to children by commands like **exec**. If the entire**en v** array is unset then Tcl will stop monitoring **env** accesses and will not update environment variables.

Under Windows, the environment variables PATH and COMSPEC in any capitalization are converted automatically to upper case. For instance, the PATH variable could be exported by the operating system as “path”, “Path”, “PaTh”, etc., causing otherwise simple Tcl code to have to support many special cases. All other environment variables inherited by Tcl are left unmodified. Setting an env array variable to blank is the same as unsetting it as this is the behavior of the underlying Windows OS. It should be noted that relying on an existing and empty environment variable will not work on Windows and is discouraged for cross-platform usage.

The following elements of **env** are special to Tcl:

env(HOME)

This environment variable, if set, gives the location of the directory considered to be the current user’s home directory, and to which a call of **cd** without arguments or with just “~” as an argument will change into. Most platforms set this correctly by default; it does not normally need to be set by user code.

env(TCL_LIBRARY)

If set, then it specifies the location of the directory containing library scripts (the value of this variable will be assigned to the **tcl_library** variable and therefore returned by the command **info library**). If this variable is not set then a default value is used.

Note that this environment variable should *not* normally be set.

env(TCLLIBPATH)

If set, then it must contain a valid Tcl list giving directories to search during auto-load operations. Directories must be specified in Tcl format, using “/” as the path separator, regardless of platform. This variable is only used when initializing the **auto_path** variable.

env(TCL_TZ), **env(TZ)**
These specify the default timezone used for parsing and formatting times and dates in the **clock** command. On many platforms, the TZ environment variable is set up by the operating system.

env(LC_ALL), **env(LC_MESSAGES)**, **env(LANG)**
These environment variables are used by the **msgcat** package to determine what locale to format messages using.

env(TCL_INTERP_DEBUG_FRAME)
If existing, it has the same effect as running **interp debug {} -frame 1** as the very first command of each new Tcl interpreter.

errorCode
This variable holds the value of the **−errorcode** return option set by the most recent error that occurred in this interpreter. This list value represents additional information about the error in a form that is easy to process with programs. The first element of the list identifies a general class of errors, and determines the format of the rest of the list. The following formats for **−errorcode** return options are used by the Tcl core; individual applications may define additional formats.

ARITH *code msg*
This format is used when an arithmetic error occurs (e.g. an attempt to divide zero by zero in the **expr** command). *Code* identifies the precise error and *msg* provides a human-readable description of the error. *Code* will be either DIVZERO (for an attempt to divide by zero), DOMAIN (if an argument is outside the domain of a function, such as acos(−3)), IOVERFLOW (for integer overflow), OVERFLOW (for a floating-point overflow), or UNKNOWN (if the cause of the error cannot be determined).

Detection of these errors depends in part on the underlying hardware and system libraries.

CHILDKILLED *pid sigName msg*
This format is used when a child process has been killed because of a signal. The *pid* element will be the process's identifier (in decimal). The *sigName* element will be the symbolic name of the signal that caused the process to terminate; it will be one of the names from the include file signal.h, such as **SIGPIPE**. The*msg* element will be a short human-readable message describing the signal, such as "write on pipe with no readers" for **SIGPIPE**.

CHILDSTATUS *pid code*
This format is used when a child process has exited with a non-zero exit status. The *pid* element will be the process's identifier (in decimal) and the *code* element will be the exit code returned by the process (also in decimal).

CHILDSUSP *pid sigName msg*
This format is used when a child process has been suspended because of a signal. The *pid* element will be the process's identifier, in decimal. The *sigName* element will be the symbolic name of the signal that caused the process to suspend; this will be one of the names from the include file signal.h, such as **SIGTTIN**. The*msg* element will be a short human-readable message describing the signal, such as "background tty read" for **SIGTTIN**.

NONE This format is used for errors where no additional information is available for an error besides the message returned with the error. In these cases the**−err orcode** return option will consist of a list containing a single element whose contents are **NONE**.

POSIX *errName msg*
If the first element is **POSIX**, then the error occurred during a POSIX kernel call. The *errName* element will contain the symbolic name of the error that occurred, such as **ENOENT**; this will be one of the values defined in the include file errno.h. The *msg*

element will be a human-readable message corresponding to *errName*, such as "no such file or directory" for the **ENOENT** case.

TCL ... Indicates some sort of problem generated in relation to Tcl itself, e.g. a failure to look up a channel or variable.

To set the **−errorcode** return option, applications should use library procedures such as **Tcl_SetObjErrorCode**, **Tcl_SetReturnOptions**, and **Tcl_PosixError**, or they may invoke the **−errorcode** option of the **return** command. If none of these methods for setting the error code has been used, the Tcl interpreter will reset the variable to **NONE** after the next error.

errorInfo

This variable holds the value of the **−errorinfo** return option set by the most recent error that occurred in this interpreter. This string value will contain one or more lines identifying the Tcl commands and procedures that were being executed when the most recent error occurred. Its contents take the form of a stack trace showing the various nested Tcl commands that had been invoked at the time of the error.

tcl_library

This variable holds the name of a directory containing the system library of Tcl scripts, such as those used for auto-loading. The value of this variable is returned by the **info library** command. See the **library** manual entry for details of the facilities provided by the Tcl script library. Normally each application or package will have its own application-specific script library in addition to the Tcl script library; each application should set a global variable with a name like **$*app*_library** (where *app* is the application's name) to hold the network file name for that application's library directory. The initial value of **tcl_library** is set when an interpreter is created by searching several different directories until one is found that contains an appropriate Tcl startup script. If the **TCL_LIBRARY** environment variable exists, then the directory it names is checked first. If **TCL_LIBRARY** is not set or doesn't refer to an appropriate directory, then Tcl checks several other directories based on a compiled-in default location, the location of the binary containing the application, and the current working directory.

tcl_patchLevel

When an interpreter is created Tcl initializes this variable to hold a string giving the current patch level for Tcl, such as **8.4.16** for Tcl 8.4 with the first sixteen official patches, or **8.5b3** for the third beta release of Tcl 8.5. The value of this variable is returned by the **info patchlevel** command.

tcl_pkgPath

This variable holds a list of directories indicating where packages are normally installed. It is not used on Windows. It typically contains either one or two entries; if it contains two entries, the first is normally a directory for platform-dependent packages (e.g., shared library binaries) and the second is normally a directory for platform-independent packages (e.g., script files). Typically a package is installed as a subdirectory of one of the entries in the **tcl_pkgPath** variable. The directories in the **tcl_pkgPath** variable are included by default in the **auto_path** variable, so they and their immediate subdirectories are automatically searched for packages during **package require** commands. Note:**tcl_pkgP ath** is not intended to be modified by the application. Its value is added to **auto_path** at startup; changes to **tcl_pkgPath** are not reflected in **auto_path**. If you want Tcl to search additional directories for packages you should add the names of those directories to **auto_path**, not **tcl_pkgPath**.

tcl_platform

This is an associative array whose elements contain information about the platform on which the application is running, such as the name of the operating system, its current release number, and the machine's instruction set. The elements listed below will always be defined, but they may have empty strings as values if Tcl could not retrieve any relevant information. In addition, extensions and applications may add additional values to the array. The predefined elements are:

byteOrder
The native byte order of this machine: either **littleEndian** or **bigEndian**.

debug If this variable exists, then the interpreter was compiled with and linked to a debug-enabled C run-time. This variable will only exist on Windows, so extension writers can specify which package to load depending on the C run-time library that is in use. This is not an indication that this core contains symbols.

engine The name of the Tcl language implementation. When the interpreter is first created, this is always set to the string **Tcl**.

machine
The instruction set executed by this machine, such as **intel**, **PPC**, **68k**, or **sun4m**. On UNIX machines, this is the value returned by **uname -m**.

os The name of the operating system running on this machine, such as **Windows NT** or **SunOS**. On UNIX machines, this is the value returned by **uname -s**.

osVersion
The version number for the operating system running on this machine. On UNIX machines, this is the value returned by **uname -r**.

pathSeparator
The character that should be used to **split** PATH-like environment variables into their corresponding list of directory names.

platform
Either **windows**, or **unix**. This identifies the general operating environment of the machine.

pointerSize
This gives the size of the native-machine pointer in bytes (strictly, it is same as the result of evaluating *sizeof(void*)* in C.)

threaded
If this variable exists, then the interpreter was compiled with threads enabled.

user This identifies the current user based on the login information available on the platform. This value comes from the getuid() and getpwuid() system calls on Unix, and the value from the GetUserName() system call on Windows.

wordSize
This gives the size of the native-machine word in bytes (strictly, it is same as the result of evaluating *sizeof(long)* in C.)

tcl_precision
This variable controls the number of digits to generate when converting floating-point values to strings. It defaults to 0. *Applications should not c hange this value;* it is provided for compatibility with legacy code.

The default value of 0 is special, meaning that Tcl should convert numbers using as few digits as possible while still distinguishing any floating point number from its nearest neighbours. It differs from using an arbitrarily high value for *tcl_precision* in that an inexact number like *1.4* will convert as *1.4* rather than *1.3999999999999999* even though the latter is nearer to the exact value of the binary number.

If **tcl_precision** is not zero, then when Tcl converts a floating point number, it creates a decimal representation of at most **tcl_precision** significant digits; the result may be shorter if the shorter result represents the original number exactly. If no result of at most **tcl_precision** digits is an exact representation of the original number, the one that is closest to the original number is chosen. If the original number lies precisely between two equally accurate decimal representations, then the

one with an even value for the least significant digit is chosen; for instance, if **tcl_precision** is 3, then 0.3125 will convert to 0.312, not 0.313, while 0.6875 will convert to 0.688, not 0.687. Any string of trailing zeroes that remains is trimmed.

a **tcl_precision** value of 17 digits is "perfect" for IEEE floating-point in that it allows double-precision values to be converted to strings and back to binary with no loss of information. For this reason, you will often see it as a value in legacy code that must run on Tcl versions before 8.5. It is no longer recommended; as noted above, a zero value is the preferred method.

All interpreters in a thread share a single **tcl_precision** value: changing it in one interpreter will affect all other interpreters as well. Safe interpreters are not allowed to modify the variable.

Valid values for **tcl_precision** range from 0 to 17.

tcl_rcFileName

This variable is used during initialization to indicate the name of a user-specific startup file. If it is set by application-specific initialization, then the Tcl startup code will check for the existence of this file and **source** it if it exists. For example, for **wish** the variable is set to **~/.wishrc** for Unix and **~/wishrc.tcl** for Windows.

tcl_traceCompile

The value of this variable can be set to control how much tracing information is displayed during bytecode compilation. By default, **tcl_traceCompile** is zero and no information is displayed. Setting **tcl_traceCompile** to 1 generates a one-line summary in **stdout** whenever a procedure or top-level command is compiled. Setting it to 2 generates a detailed listing in **stdout** of the bytecode instructions emitted during every compilation. This variable is useful in tracking down suspected problems with the Tcl compiler.

This variable and functionality only exist if **TCL_COMPILE_DEBUG** was defined during Tcl's compilation.

tcl_traceExec

The value of this variable can be set to control how much tracing information is displayed during bytecode execution. By default, **tcl_traceExec** is zero and no information is displayed. Setting **tcl_traceExec** to 1 generates a one-line trace in **stdout** on each call to a Tcl procedure. Setting it to 2 generates a line of output whenever any Tcl command is invoked that contains the name of the command and its arguments. Setting it to 3 produces a detailed trace showing the result of executing each bytecode instruction. Note that when **tcl_traceExec** is 2 or 3, commands such as **set** and **incr** that have been entirely replaced by a sequence of bytecode instructions are not shown. Setting this variable is useful in tracking down suspected problems with the bytecode compiler and interpreter.

This variable and functionality only exist if **TCL_COMPILE_DEBUG** was defined during Tcl's compilation.

tcl_wordchars

The value of this variable is a regular expression that can be set to control what are considered "word" characters, for instances like selecting a word by double-clicking in text in Tk. It is platform dependent. On Windows, it defaults to **\S**, meaning anything but a Unicode space character. Otherwise it defaults to **\w**, which is any Unicode word character (number, letter, or underscore).

tcl_nonwordchars

The value of this variable is a regular expression that can be set to control what are considered "non-word" characters, for instances like selecting a word by double-clicking in text in Tk. It is platform dependent. On Windows, it defaults to **\s**, meaning any Unicode space character. Otherwise it defaults to **\W**, which is anything but a Unicode word character (number, letter, or underscore).

tcl_version
When an interpreter is created Tcl initializes this variable to hold the version number for this version of Tcl in the form *x.y*. Changes to*x* represent major changes with probable incompatibilities and changes to *y* represent small enhancements and bug fixes that retain backward compatibility. The value of this variable is returned by the **info tclversion** command.

OTHER GLOBAL VARIABLES

The following variables are only guaranteed to exist in **tclsh** and **wish** executables; the Tcl library does not define them itself but many Tcl environments do.

argc The number of arguments to **tclsh** or **wish**.

argv Tcl list of arguments to **tclsh** or **wish**.

argv0 The script that **tclsh** or **wish** started executing (if it was specified) or otherwise the name by which **tclsh** or **wish** was invoked.

tcl_interactive
Contains 1 if **tclsh** or **wish** is running interactively (no script was specified and standard input is a terminal-like device), 0 otherwise.

EXAMPLES

To add a directory to the collection of locations searched by **package require**, e.g., because of some application-specific packages that are used, the **auto_path** variable needs to be updated:

```
lappend ::auto_path [file join [pwd] "theLibDir"]
```

A simple though not very robust way to handle command line arguments of the form “−foo 1 −bar 2” is to load them into an array having first loaded in the default settings:

```
array set arguments {-foo 0 -bar 0 -grill 0}
array set arguments $::argv
puts "foo is $arguments(-foo)"
puts "bar is $arguments(-bar)"
puts "grill is $arguments(-grill)"
```

The **argv0** global variable can be used (in conjunction with the **info script** command) to determine whether the current script is being executed as the main script or loaded as a library. This is useful because it allows a single script to be used as both a library and a demonstration of that library:

```
if {$::argv0 eq [info script]} {
   # running as: tclsh example.tcl
} else {
   package provide Example 1.0
}
```

SEE ALSO

eval(n), library(n), tclsh(1), tkvars(n), wish(1)

KEYWORDS

arithmetic, bytecode, compiler, error, environment, POSIX, precision, subprocess, user, variables