NAME

catch − Evaluate script and trap exceptional returns

SYNOPSIS

catch *script* ?*resultVarName*? ?*optionsVarName*?

DESCRIPTION

The **catch** command may be used to prevent errors from aborting command interpretation. The **catch** command calls the Tcl interpreter recursively to execute *script*, and always returns without raising an error, regardless of any errors that might occur while executing *script*.

If *script* raises an error, **catch** will return a non-zero integer value corresponding to the exceptional return code returned by evaluation of *script*. Tcl defines the normal return code from script evaluation to be zero (0), or **TCL_OK**. Tcl also defines four exceptional return codes: 1 (**TCL_ERROR**), 2 (**TCL_RETURN**), 3 (**TCL_BREAK**), and 4 (**TCL_CONTINUE**). Errors during evaluation of a script are indicated by a return code of **TCL_ERROR**. The other exceptional return codes are returned by the **return**, **break**, and **continue** commands and in other special situations as documented. Tcl packages can define new commands that return other integer values as return codes as well, and scripts that make use of the **return −code** command can also have return codes other than the five defined by Tcl.

If the *resultVarName* argument is given, then the variable it names is set to the result of the script evaluation. When the return code from the script is 1 (**TCL_ERROR**), the value stored in *resultVarName* is an error message. When the return code from the script is 0 (**TCL_OK**), the value stored in *resultVarName* is the value returned from *script*.

If the *optionsVarName* argument is given, then the variable it names is set to a dictionary of return options returned by evaluation of *script*. Tcl specifies two entries that are always defined in the dictionary: **−code** and **−level**. When the return code from evaluation of *script* is not **TCL_RETURN**, the value of the **−level** entry will be 0, and the value of the **−code** entry will be the same as the return code. Only when the return code is **TCL_RETURN** will the values of the **−level** and **−code** entries be something else, as further described in the documentation for the **return** command.

When the return code from evaluation of *script* is **TCL_ERROR**, four additional entries are defined in the dictionary of return options stored in *optionsVarName*: **−errorinfo**, **−errorcode**, **−errorline**, and **−errorstack**. The value of the **−errorinfo** entry is a formatted stack trace containing more information about the context in which the error happened. The formatted stack trace is meant to be read by a person. The value of the **−errorcode** entry is additional information about the error stored as a list. The **−errorcode** value is meant to be further processed by programs, and may not be particularly readable by people. The value of the **−errorline** entry is an integer indicating which line of *script* was being evaluated when the error occurred. The value of the **−errorstack** entry is an even-sized list made of token-parameter pairs accumulated while unwinding the stack. The token may be "**CALL**", in which case the parameter is a list made of the proc name and arguments at the corresponding level; or it may be "**UP**", in which case the parameter is the relative level (as in **uplevel**) of the previous **CALL**. The salient differences with respect to **−errorinfo** are that:

[1] it is a machine-readable form that is amenable to processing with [**foreach** {tok prm} ...],

[2] it contains the true (substituted) values passed to the functions, instead of the static text of the calling sites, and

[3] it is coarser-grained, with only one element per stack frame (like procs; no separate elements for **foreach** constructs for example).

The values of the **−errorinfo** and **−errorcode** entries of the most recent error are also available as values of the global variables **::errorInfo** and **::errorCode** respectively. The value of the **−errorstack** entry surfaces as **info errorstack**.

Tcl packages may provide commands that set other entries in the dictionary of return options, and the

return command may be used by scripts to set return options in addition to those defined above.

EXAMPLES

The **catch** command may be used in an **if** to branch based on the success of a script.

```
if { [catch {open $someFile w} fid] } {
    puts stderr "Could not open $someFile for writing\n$fid"
    exit 1
}
```

There are more complex examples of **catch** usage in the documentation for the **return** command.

SEE ALSO

break(n), continue(n), dict(n), error(n), errorCode(n), errorInfo(n), info(n), return(n)

KEYWORDS

catch, error, exception