

Tcl Variable Utilities

Version 0.3

Alex Baker

<https://github.com/ambaker1/vutil>

June 2, 2023

Abstract

This package provides various utilities for working with variables in Tcl.

Printing Variables to Screen

The *pvar* command is a short-hand function for printing the name and values of Tcl variables.

```
pvar $name1 $name2 ...
```

`$name1 $name2 ...` Name(s) of variables to print

Example 1: Printing variables to screen

Code:

```
set a 5
set b 7
set c(1) 5
set c(2) 6
pvar a b c
```

Output:

```
a = 5
b = 7
c(1) = 5
c(2) = 6
```

Initializing Local Namespace Variables

The command *local* is the counterpart to the Tcl *global* command, and creates local variables linked to variables in the current namespace, by simply calling the Tcl *variable* command multiple times.

```
local $name1 $name2 ...
```

`$name1 $name2 ...` Name(s) of variables to initialize

Example 2: Access namespace variables in a procedure

Code:

```
# Define global variables
global a b c
set a 1
set b 2
set c 3
namespace eval ::foo {
    # Define local variables
    local a b c
    set a 4
    set b 5
    set c 6
}
proc ::foo::bar1 {} {
    # Access global variables
    global a b c
    list $a $b $c
}
proc ::foo::bar2 {} {
    # Access local variables
    local a b c
    list $a $b $c
}
puts [::foo::bar1]; # global a b c
puts [::foo::bar2]; # local a b c
```

Output:

```
1 2 3
4 5 6
```

Default Values

The command *default* assigns values to variables if they do not exist.

```
default $varName $value
```

\$varName Name of variable to set

\$value Default value for variable

The example below shows how default values are only applied if the variable does not exist.

Example 3: Variable defaults

Code:

```
set a 5
default a 7
puts $a
unset a
default a 7
puts $a
```

Output:

```
5
7
```

Variable Locks

The command *lock* uses Tcl variable traces to make a read-only variable.

```
lock $varName <$value>
```

\$varName Variable name to lock.

\$value Value to lock variable at. Default self-locks (uses current value).

The command *unlock* unlocks previously locked variables so that they can be modified again.

```
unlock $name1 $name2 ...
```

\$name1 \$name2 ... Variables to unlock.

Example 4: Variable locks

Code:

```
lock a 5
set a 7
puts $a
unlock a
set a 7
puts $a
```

Output:

```
5
7
```

Variable-Object Ties

As of Tcl version 8.6, there is no garbage collection for Tcl objects, they have to be removed manually with the “destroy” method. The command *tie* is a solution for this problem, using variable traces to destroy the corresponding object when the variable is unset or modified. Tie is separate from lock; a tie will override a lock, and a lock will override a tie.

```
tie $varName <$object>
```

\$varName Variable name to tie to object.

\$object Object to tie variable to. Default self-ties (uses current value).

In similar fashion to *unlock*, tied variables can be untied with the command *untie*.

```
untie $name1 $name2 ...
```

\$name1 \$name2 ... Variables to untie.

Example 5: Variable-object ties

Code:

```
oo::class create foo {
    method hi {} {
        puts hi
    }
}
tie a [foo create bar]
set b $a; # alias variable
unset a; # triggers ``destroy''
$b hi; # throws error
```

Output:

```
invalid command name "::bar"
```