Flytrap: Tcl Debugging Tools

Version 1.1.1

Alex Baker

https://github.com/ambaker1/flytrap

March 12, 2025

Abstract

Say goodbye to debugging with countless *puts* statements, and say hello to "flytrap"!

Pausing a Script

The *pause* command pauses a Tcl script, prints the file and line number, and enters command-line mode, allowing the user to query variables and insert code into an analysis. If the command entered while paused returns an error, the error message will be displayed and the script will remain paused. If the command entered is "return", the pause will be exited and the corresponding result and options will be passed to the caller. For example, a loop can be broken by entering *return -code break* in pause mode. Pressing enter with no commands will simply continue the script.

```
pause <$frameOffset>
```

$frameOffset	Frame offset for calling pause in another level (e.g. uplevel) Default 0.

Example 1: Pausing an analysis

Code:

```
pause
```

Output:

```
PAUSED...
(line 407 file "C:/User/Documents/MyFile.tcl")
>
```

Note: If in interactive mode, there may not be a file to pause in. In this case, it will list the procedure or script where the pause occurred.

Advanced Tcl Debugger

The *flytrap* command parses a Tcl script, and prints out the evaluation steps and results if an error is reached. Additionally, if an error is reached, the script will pause at the line where the error occurred, allowing for interactive introspection of the problem, at the depth specified.

```
flytrap <-depth $maxDepth> <-verbose $verboseFlag> (-file $filename | <-body> $body)
```

$maxDepth	Optional recursive depth to step into procedures (default 0).
$verboseFlag	Optional flag to always print out all steps and results (default 0).
$filename	File path of Tcl script to debug. Mutually exclusive with -body.
$body	Tcl script to debug. Mutually exclusive with -file.

Example 2: Verbose evaluation of a procedure

Code:

```
proc add {a b} {
    return [expr {$a + $b}]
}
set a 5
set b 7
flytrap -depth 1 -verbose true -body {
    add [expr {$a*2}] $b
}
```

Output:

```
> expr {$a*2}
10
> add 10 7
  > expr {$a + $b}
  17
  > return 17
  17
17
```

Printing Variables to Screen

The *printVars* command is a short-hand function for printing the name and values of Tcl variables, in the same style as the Tcl *parray* command.

```
printVars $name1 $name2 ...
```

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

Example 3: Printing variables to screen

Code:

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

Output:

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

Variable Viewer Widget Class

The class *varViewer* is a TclOO class that creates widget objects that display the values of variables. It can be used to monitor variable values in a widget.

```
varViewer new $varList <$title>
varViewer create $name $varList <$title>
```

$name	Object name.
$varList	List of variables to view.
$title	Optional title. Default "Workspace".

Example 4: Monitoring variable values

Code:

```
set i 0
varViewer new i {counter}
for {set i 0} {$i < 1000} {incr i} {
    update
}
```

The command *viewVars* opens up a *varViewer* widget displaying the values of all the variables in the current scope, and then pauses the script using the *pause* command, such that continuing destroys the widget.

```
viewVars
```

Example 5: Workspace viewer

Code:

```
set a 5
set b 7
array set c {1 5 2 6}
viewVars
```

Output: