Flytrap: Tcl Debugging Tools

Version 0.1.2

Alex Baker

https://github.com/ambaker1/flytrap

May 24, 2023

Abstract

Since OpenSees is a script-based finite element analysis software, it can be difficult to debug a model or analysis when problems arise. Typically, *puts* statements are the extent of debugging a script written in Tcl, but this method can be cumbersome for more complex scripts. Towards making OpenSees Tcl more user-friendly, the "flytrap" package makes debugging code easy.

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 -file $filename <$depth> <$verbose>
flytrap -body $script <$depth> <$verbose>
```

$filename	File path of Tcl script to debug.
$script	Tcl script to debug.
$depth	Optional recursive depth to step into procedures (default 0).
$verbose	Optional flag to always print out all steps and results (default 0).

Example 1: Verbose evaluation of a procedure

Code:

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

Output:

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

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

Example 2: 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.

Unit Testing

The command *assert* can be used for basic unit testing of Tcl scripts. It throws an error if the statement is false. If the statement is true, it simply returns nothing and the script continues.

```
assert $value1 <$op $value2>
```

$value1	Value to test.
$op	Comparison operator. Default "==".
$value2	Comparison value. Default 1, or "true".

Example 3: Validation for unit testing

Code:

```
assert [string is double 5.0]; # Asserts that 5.0 is a number
assert [expr {2 + 2}] == 4; # Asserts that math works
```

Printing Variables to Screen

The *pvar* command is a short-hand function for printing the name and value of Tcl variables, including arrays.

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

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

Example 4: Printing variables to screen

Code:

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

Output:

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

Interactive Workspace Viewer

The command *viewVars* pauses a Tcl script and opens up an interactive table of all variables in the current scope and their values. Variables cannot be edited in the variable viewer window, but their values can be selected and copied. This command in particular requires the packages "Tk" and "Tktable".

```
viewVars <$var1 $var2 ...>
```

`$var1 $var2 ...`	Variables to view. Default all in current scope (minus tclvars).

Example 5: Workspace viewer

Code:

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

Output: