



Keeping Locals Local or Prophylaxis for Power Files

Judith Richardson
judith.richardson@nxp.com

NXP Semiconductors, Corporate I&T, Design Technology and Flows
October 25, 2007



The Disease

UPF files are scripts of Tcl commands

Some script writers use the full power of Tcl, using local and global variables

Frequently two scripts will use the same variable names

Is this the same variable?

There can be different effects if the files are “sourced” or loaded with the load_upf command

source - all variables can be accessed in the file being read. They will keep their changed values after loading.

load_upf - which variables can be accessed in the file being read? Which will keep their changed values after loading?



The Symptoms - source

```
if { ![info exists localVariable] } {  
    puts "localVariable NA"  
} else {  
    puts " localVariable $localVariable"  
}  
set localVariable 3  
global globalVariable  
if { ![info exists globalVariable] } {  
    puts "globalVariable NA"  
} else {  
    puts "globalVariable $globalVariable"  
}  
set globalVariable 4  
global globalChildVariable  
set globalChildVariable 11  
set childVariable 10
```



The Symptoms - source

```
if { ![info exists localVariable] } {  
    puts "localVariable NA"  
} else {  
    puts " localVariable $localVariable"  
}  
set localVariable 3
```



The Symptoms - source

```
global globalVariable
```

```
if { ![info exists globalVariable] } {
```

```
    puts "globalVariable NA"
```

```
} else {
```

```
    puts "globalVariable $globalVariable"
```

```
}
```

```
set globalVariable 4
```



The Symptoms - source
global globalChildVariable
set globalChildVariable 11
set childVariable 10



The Symptoms - source

```
if { ![info exists localVariable] } {  
  puts "localVariable NA"  
} else {  
  puts " localVariable $localVariable"  
}  
set localVariable 3  
global globalVariable  
if { ![info exists globalVariable] } {  
  puts "globalVariable NA"  
} else {  
  puts "globalVariable $globalVariable"  
}  
set globalVariable 4  
global globalChildVariable  
set globalChildVariable 11  
set childVariable 10
```

Source from another file

```
global globalVariable  
set globalVariable 2  
set localVariable 1  
source child_variables.upf
```

Result

In child

```
localVariable 1  
globalVariable 2
```

In calling file

```
localVariable 3  
globalVariable 4  
childVariable 10  
globalChildVariable 11
```



The Symptoms – load_upf

```
if{![info exists localVariable]}{puts "localVariable NA"  
  }else{puts " localVariable $localVariable"}  
set localVariable 3  
global globalVariable  
if{![info exists globalVariable]}{puts "globalVariable NA"  
  }else{puts "globalVariable $globalVariable"}  
set globalVariable 4  
global globalChildVariable  
set globalChildVariable 11  
set childVariable 10
```

source result

In child

localVariable 1
globalVariable 2

In calling file

localVariable 3
globalVariable 4
childVariable 10
globalChildVariable 11

Load_upf from another file

```
global globalVariable  
set globalVariable 2  
set localVariable 1  
load_upf child_variables.upf
```

load_upf Result

In child

localVariable NA
globalVariable NA

In calling file

localVariable 1
globalVariable 2
childVariable NA
globalChildVariable NA



The Prophylactic - load_protected_upf

Define a new command that provides a controlled mechanism for passing values between files while protecting global variables

Usage :

`load_protected_upf fileName [-hide_globals] [-scope scopeName] [-version upfVersion] [-params paramList]`

`-hide_globals` will save all globals before sourcing the `fileName`, and then restore them afterwards. Globals named in the `-params` list will keep any modified value resulting from sourcing the file. Any globals created in the sourced file, other than the ones named in `params`, will be unset at the end of loading.

`-params` provides a list of variables to be made available while sourcing the file.

In `paramList` each element is either

`paramName`

in this case `paramName` will be declared as "global \$paramName". Any changes made to this variable will be visible at the calling level after this command completes

or

`{paramName paramValue}`

in this case a local variable `paramName` will be created and its initial value set to `paramValue`

Other parameters as for `load_upf`



The Prophylactic - load_protected_upf

-hide_globals will save all globals before sourcing the fileName, and then restore them afterwards. Globals named in the -params list will keep any modified value resulting from sourcing the file. Any globals created in the sourced file, other than the ones named in -params will be unset at the end of loading.

The Prophylactic - load_protected_upf

-params provides a list of variables to be made available while sourcing the file.

In paramList each element is either
paramName

in this case paramName will be declared as "global \$paramName". Any changes made to this variable will be visible at the calling level after this command completes

or

{paramName paramValue}

in this case a local variable paramName will be created and its initial value set to paramValue



Prevention in action - 1

Protect everything

load_protected_upf -hide_globals child_variables.upf

In child

localVariable NA globalVariable NA

In calling file

localVariable 1 globalVariable 2
childVariable NA globalChildVariable NA

source result

In child

localVariable 1 globalVariable 2

In calling file

localVariable 3 globalVariable 4
childVariable 10 globalChildVariable 11

load_upf result

In child

localVariable NA globalVariable NA

In calling file

localVariable 1 globalVariable 2
childVariable NA globalChildVariable NA



Prevention in action - 2

Pass variable into called file

```
load_protected_upf -hide_globals \  
-params {{localVariable 6}} child_variables.upf
```

In child

localVariable 6 globalVariable NA

In calling file

localVariable 1 globalVariable 2
childVariable NA globalChildVariable NA

source result

In child

localVariable 1 globalVariable 2

In calling file

localVariable 3 globalVariable 4
childVariable 10 globalChildVariable 11

load_upf result

In child

localVariable NA globalVariable NA

In calling file

localVariable 1 globalVariable 2
childVariable NA globalChildVariable NA



Prevention in action - 3

Pass variable into called file and all globals visible

**load_protected_upf -params {{localVariable 6}} \
child_variables.upf**

In child

localVariable 6 globalVariable 2

In calling file

**localVariable 1 globalVariable 4
childVariable NA globalChildVariable 11**

source result

In child

localVariable 1 globalVariable 2

In calling file

**localVariable 3 globalVariable 4
childVariable 10 globalChildVariable 11**

load_upf result

In child

localVariable NA globalVariable NA

In calling file

**localVariable 1 globalVariable 2
childVariable NA globalChildVariable NA**



Prevention in action - 4

Pass variable into called file, only some globals visible

`load_protected_upf -hide_globals -params \`
`{globalVariable {localVariable 6}} child_variables.upf`

In child

localVariable 6 globalVariable 2

In calling file

localVariable 1 globalVariable 4
childVariable NA globalChildVariable NA

source result

In child

localVariable 1 globalVariable 2

In calling file

localVariable 3 globalVariable 4
childVariable 10 globalChildVariable 11

load_upf result

In child

localVariable NA globalVariable NA

In calling file

localVariable 1 globalVariable 2
childVariable NA globalChildVariable NA



Prevention in action - 5

Only specific child globals visible in calling file

`load_protected_upf -hide_globals -params \`
`{globalChildVariable} child_variables.upf`

In child

localVariable NA globalVariable NA

In calling file

localVariable 1 globalVariable 2
childVariable NA globalChildVariable 11

source result

In child

localVariable 1 globalVariable 2

In calling file

localVariable 3 globalVariable 4
childVariable 10 globalChildVariable 11

load_upf result

In child

localVariable NA globalVariable NA

In calling file

localVariable 1 globalVariable 2
childVariable NA globalChildVariable NA



The Outcome

The user has full control over which variables are visible in both the calling file and the called file

No unexpected side-effects





Implementation

```
#  
# Demonstration implementation of load_protected_upf  
#  
# Purpose :  
# Load a UPF file in a protected environment that prevents corruption of  
# existing variables  
#  
# Usage :  
# load_protected_upf fileName \  
#     [-hide_globals] \  
#     [-scope scopeName] \  
#     [-version upfVersion] \  
#     [-params paramList]  
#  
# fileName UPF file to be sourced  
# -hide_globals will save all globals before sourcing the file fileName,  
# and then restore them afterwards. Globals named in the -params list will  
# keep any modified value resulting from sourcing the file. Any globals  
# created in the sourced file, other than the ones named in params, will  
# be unset at the end of loading.  
# -scope sets the scope where the file will be sourced  
# -version sets the UPF version to be used for the commands read from the  
# file  
# -params provides a list of variables to be made available while sourcing the  
# file.  
# In paramList each element is either  
# paramName  
#     in this case paramName will be declared as "global $paramName".  
#     Any changes made to this variable will be visible at the calling  
#     level after this command completes  
# or  
# {paramName paramValue}  
#     in this case a local variable paramName will be created and its  
#     initial value set to paramValue  
#  
# Return value 0 if command failed, 1 if successful  
#
```



Implementation

```
#  
# Requires global variable hierarchySeparator  
#  
# Relies on local variables prefixed with load_protected_ not being corrupted in the sourced file  
#  
# Calls UPF commands :  
# set_scope  
# upf_version  
# Calls local commands (defined in this file) :  
# save_globals  
# restore_globals  
#  
# Author :  
# Judith Richardson, NXP Semiconductors  
#
```



Implementation

```
proc load_protected_upf args {
  global hierarchySeparator
  #
  # Default is to allow globals to be modified
  #
  set load_protected_hideGlobals 0
  #
  # Parse the command arguments
  #
  for {set i 0} {$i < [length $args]} {incr i 1} {
    set arg [lindex $args $i]
    #
    # Handle options
    #
    if { [string index $arg 0] == "-" } {
      if { [string match "$arg*" "-version" ] } {
        incr i 1
        set load_protected_version [lindex $args $i]
      } else {
        if { [string match "$arg*" "-scope" ] } {
          incr i 1
          set load_protected_scope [lindex $args $i]
        } else {
          if { [string match "$arg*" "-params" ] } {
            incr i 1
            set load_protected_params [lindex $args $i]
          } else {
            if { [string match "$arg*" "-hide_globals" ] } {
              set load_protected_hideGlobals 1
            } else {
              puts "Error : load_protected_upf : Unrecognised option $arg to load_protected_upf"
              return 0
            }
          }
        }
      }
    }
  }
}
} else {
```



Implementation

```
#
# There must be exactly one fileName argument
#
if { [info exists load_protected_fileName] } {
    puts "Error : load_protected_upf : File name $load_protected_fileName already specified when parsing $arg"
    return 0
}
set load_protected_fileName $arg
}
}

#
# Must specify file to load
#
if { ![info exists load_protected_fileName] } {
    puts "Error : load_protected_upf : No file name specified"
    return 0
}

#
# Problem in Tcl if the errorInfo global variable is unset
#
set load_protected_keepGlobalsList [list errorInfo]

#
# Add the names of any globals that the tool relies on
# (this implementation needs the hierarchySeparator to check for failure in set_scope)
#
lappend load_protected_keepGlobalsList hierarchySeparator
```



Implementation

```
#
# Handle any params
#
if { [info exists load_protected_params] } {
  foreach load_protected_param $load_protected_params {
    set load_protected_par [split $load_protected_param]
    if { [llength $load_protected_par] > 2 } {
      puts "Error : load_protected_upf : Bad param '$load_protected_param'"
      return 0
    }
    if { [llength $load_protected_par] == 1 } {
      #
      # Just paramName so make it global
      #
      lappend load_protected_keepGlobalsList [lindex $load_protected_par 0]
      global [lindex $load_protected_par 0]
    } else {
      #
      # paramName and paramValue so initialise local variable
      #
      set [lindex $load_protected_par 0] [lindex $load_protected_par 1]
    }
  }
}

#
# Are we protecting globals from modification ?
#
if { $load_protected_hideGlobals } {
  #
  # Save the current values of all the globals
  # Unset all the globals apart from the ones in keepGlobalsList
  #
  set load_protected_globalList [save_globals $load_protected_keepGlobalsList]
}
```



Implementation

```
#
# Set the UPF version if requested
#
set load_protected_origVersion [upf_version]
if { [info exists load_protected_version] } {
  set load_protected_origVersion [upf_version $load_protected_version]
}
#
# Set the scope if requested
#
set load_protected_origScope [set_scope .]
if { [info exists load_protected_scope] } {
  set_scope $load_protected_scope
  if { [set_scope .] == $hierarchySeparator && $load_protected_scope != $hierarchySeparator } {
    puts "Error : load_protected_upf : failed to set scope to $load_protected_scope"
    set_scope $load_protected_origScope
    return 0
  }
}
#
# Source the UPF file
#
if [catch {source $load_protected_fileName} mssg] {
  #
  # Some error detected during sourcing the file
  #
  puts "Error : load_protected_upf : $mssg"
  if { $load_protected_hideGlobals } {
    #
    # Restore the global variables if we unset them
    #
    restore_globals $load_protected_globalList $load_protected_keepGlobalsList
  }
  set_scope $load_protected_origScope
  upf_version $load_protected_origVersion
  return 0
}
```



Implementation

```
#  
# Restore the global variables if we unset them  
#  
if { $load_protected_hideGlobals } {  
  restore_globals $load_protected_globalList $load_protected_keepGlobalsList  
}  
  
#  
# Restore the scope in case it got changed  
#  
set_scope $load_protected_origScope  
  
#  
# Restore the version in case it got changed  
#  
upf_version $load_protected_origVersion  
return 1  
}
```



Implementation

```
#  
# Demonstration implementation of save_globals  
#  
# Purpose :  
# Capture the current value of all global variables and unset them  
#  
# Usage :  
# save_globals keepGlobalsList  
#  
# keepGlobalsList list of global variable names that will not be unset  
#  
# Returns list of names of variables that were unset plus a string that can  
# be evaluated to restore its original value  
#  
# Author :  
# Judith Richardson, NXP Semiconductors  
#  
proc save_globals {save_globals_keepGlobalsList} {  
    set save_globals_globalList [list]  
    #  
    # Get list of all the global variable names  
    #  
    set save_globals_infoList [info globals]  
    for {set i 0} {$i < [llength $save_globals_infoList]} {incr i 1} {  
        set save_globals_name [lindex $save_globals_infoList $i]  
        #  
        # Skip the variable if it is in the keepGlobalsList  
        #  
        if { [lsearch $save_globals_keepGlobalsList $save_globals_name] >= 0 } {  
            continue  
        }  
  
        #  
        # Access its current value  
        #  
        global [lindex $save_globals_infoList $i]
```



Implementation

```
#
# Access its current value
#
global [lindex $save_globals_infoList $i]

if { [llength [array names $save_globals_name]] > 0 } {
#
# Array value. Convert to a list
#
set save_globals_str "array set $save_globals_name \[list [array get $save_globals_name]]"

} else {
if { [llength [set [lindex $save_globals_infoList $i]]] > 1 } {
#
# List value
#
set save_globals_str "set $save_globals_name \[list [set [lindex $save_globals_infoList $i]]]"
} else {
#
# Simple value
#
set save_globals_str "set $save_globals_name \"[set [lindex $save_globals_infoList $i]]\""
}
}
lappend save_globals_globalList [list $save_globals_name $save_globals_str]

#
# Destroy the existing global variable
#
unset $save_globals_name
}
return $save_globals_globalList
}
```



Implementation

```
#  
# Demonstration implementation of restore_globals  
#  
# Purpose :  
# Restore original values of global variables  
#  
# Usage :  
# restore_globals globalList keepGlobalsList  
#  
# globalList list of global variable names and string to be evaluated to set  
# the variables value  
# keepGlobalsList list of globals to be left untouched  
#  
# Author :  
# Judith Richardson, NXP Semiconductors  
#  
proc restore_globals {restore_globals_globalList restore_globals_keepGlobalsList} {  
# Destroy all the globals except the ones on the keepGlobalsList list  
set restore_globals_infoList [info globals]  
for {set i 0} {$i < [llength $restore_globals_infoList]} {incr i 1} {  
set restore_globals_name [lindex $restore_globals_infoList $i]  
#  
# Skip the variable if it is in the restore_globals_keepGlobalsList  
#  
if { [lsearch $restore_globals_keepGlobalsList $restore_globals_name] >= 0 } {  
continue  
}  
#  
# Access its current value  
#  
global [lindex $restore_globals_infoList $i]  
#  
# Destroy the existing global variable  
#  
unset $restore_globals_name  
}  
}
```



Implementation

```
# Re-create the original globals
foreach restore_globals_globalVariable $restore_globals_globalList {
#
# Make the variable global
#
global [lindex $restore_globals_globalVariable 0]
#
# Set its value
#
eval [lindex $restore_globals_globalVariable 1]
}
}
```



