Tcl Widget Objects (wob)

Version 1.0

Alex Baker

https://github.com/ambaker1/wob

August 14, 2023

Abstract

This package ties Tk widgets to their own TclOO objects and separate Tcl interpreters.
Additionally, "wob" provides *mainLoop*: a Tcl/Tk event loop with an interactive command line.

Creating Widget Objects

Widget objects are created from the *widget* class using the standard methods *new* or *create*. When a widget is created, it also creates a unique Tcl interpreter and loads in the Tk package, binding the "close window" event to destroy the widget object and interpreter. Once created, *widget* objects act as commands with an ensemble of subcommands, or methods. These objects can be deleted with the method *destroy*.

```
widget new <$title>
widget create $objectName <$title>
```

`$objectName`	Explicit name for object.
`$title`	Title of main widget window (default "Widget").

Example 1: Creating a widget object

Code:
```
  set widgetObj [widget new]
```

Removing Widget Objects

The standard method *destroy* removes a widget object from the main Tcl interpreter, destroying the object, widget window, and widget interpreter. Closing the widget window also destroys the widget object and interpreter.

```
$widgetObj destroy
```

Additionally, all widget objects can be closed with the command *closeAllWidgets*, or by closing the main Tcl interpreter.

```
closeAllWidgets
```

Building a Widget

All interfacing with the widget is done through its corresponding interpreter. The main method for building a widget is *eval*, which evaluates Tcl/Tk code within the widget interpreter. The method behaves the same as the Tcl *eval* command, but within the widget interpreter.

```
$widgetObj eval $arg1 $arg2 ...
```

`$arg1 $arg2 ...`   Arguments to be concatenated into a Tcl script to evaluate.

The widget's interpreter can be directly accessed with the method *interp*, for advanced introspection.

```
$widgetObj interp
```

Widget Variable Access

For convenience, variable values may be passed to the widget interpreter with the method *set*, and retrieved with *get*.

```
$widgetObj set $varName $value
```

```
$widgetObj get $varName
```

`$varName`   Name of variable in widget interpreter.

`$value`   Value to set.

Example 2: Accessing widget variables

Code:

```
set widget [widget new]
$widget set x {hello world}
puts [$widget get x]
```

Output:

```
hello world
```

Widget Variable Links

By default, variables in the widget interpreter are completely separate from the main Tcl interpreter. The method *upvar* creates a link between variables in the main interpreter and the widget interpreter so that their values are linked. If `$srcVar` does not exist, it will be initialized as blank.

```
$widgetObj upvar $srcVar $myVar <$srcVar $myVar ...>
```

`$srcVar`	Variable in parent interpreter (scalar or array element).
`$myVar`	Variable in widget interpreter (scalar or array element).

Example 3: Linking widget variables

Code:

```
set widget [widget new]
$widget upvar x x
set x {hello world}
puts [$widget get x]
```

- -

Output:

```
hello world
```

Widget Command Aliases

By default, the widget interpreter does not interface directly with the main Tcl interpreter. The method *alias* creates an alias command in the widget interpreter to access a command in the main interpreter.

```
$widgetObj alias $srcCmd $targetCmd <$arg1 $arg2 ...>
```

`$srcCmd`	Command in widget interpreter (creates the command).
`$targetCmd`	Command to link to in the main interpreter (does not create the command).
`$arg1 $arg2 ...`	Optional, prefix arguments to `$targetCmd`.

Entering the Event Loop

In order for widget components to display and be interactive, the Tk event loop must be entered. Some Tk commands automatically enter the event loop, like *tk_getOpenFile*, but for the most part, the event loop must be entered with a call to *vwait*, *tkwait*, or *update* (it is generally bad practice to use *update* though, for a variety of reasons).

The command *mainLoop* is provided as a method to enter the event loop for all widgets, while also taking interactive input from the command line, similar to the "wish.exe" Tcl/Tk program.

> **`mainLoop <$onBlank>`**

`$onBlank`   What to do after user enters a blank line: "continue" will continue the interactive event loop, and "break" will exit the interactive event loop. Default "continue".

Example 4: Entering the event loop

Code:
```
puts "Main Loop:"
mainLoop
```
- -
Output:
```
Main Loop:
> |
```

Exiting the Event Loop

To exit the event loop and continue with a script, simply enter "return" on the command line, or use the command *exitMainLoop*, which can also be scheduled as an event with the Tcl *after* command.

> **`exitMainLoop <$option $value ...> <$result>`**

`$option $value ...`   Tcl *return* options.

`$result`   Value to pass as result of *mainLoop*.

Basic Applications

The example below demonstrates how the wob package can be used to create and manipulate Tk widgets.

Example 5: Filename dialog

Code:

```
set widget [widget new]
set filename [$widget eval tk_getOpenFile]
$widget destroy
puts $filename
```

Example 6: Option selection

Code:

```
set widget [widget new]
$widget eval {
        label .label -text "Choose analysis type:"
        tk_optionMenu .options AnalysisType "" Pushover Dynamic
        pack .label -side top -fill x
        pack .options -side bottom -fill x
        vwait AnalysisType
}
puts [$widget get AnalysisType]
$widget destroy
```

Example 7: Access clipboard

Code:

```
set widget [widget new]
$widget set text "hello world"
$widget eval {
        clipboard clear
        clipboard append $text
}
mainLoop
# now the text "hello world" can be pasted into another application.
```