Adding a toolkit to gWidgets

John Verzani, gWidgetsRGtk@gmail.com

October 12, 2006

Abstract:

This little vignette illustrates what is required to write a toolkit for the **gWidgets** package. Since the **gWidgetsRGtk** package is written this sketches out what a toolkit would possible look like using the **tcltk** package. Unfortunately, this author does not know enough about the **tcltk** package to actually do this.

Contents

1 Basics of gWidgets

The gWidgets implementation is simply a set of functions that dispatch to similarly named functions in a toolkit. That is the `glabel(...,toolkit=guiToolkit())` function dispatches to the `.glabel(toolkit, ...)` function in the appropriate toolkit, and the `svalue(obj,...)` method dispatches to the `.svalue(obj@widget,obj@toolkit, ...)` function in the appropriate toolkit. In the first case the dispatch is done by the class of the toolkit. For the method, the dispatch is based on both the toolkit and the class of the object, and perhaps other arguments in the signature of the method.

As such, the basic structure of gWidgets is to set up some classes, most notable a class `guiWidgetsToolkit` of which each toolkit class is a subclass, and a set of methods for dispatch.

2 An example

As **gWidgets** is supposed to be cross-toolkit, it would be nice were there a toolkit implementation for the **tcltk** package. I know only as much about **tcltk** as was learned by browsing P. Dalgaard's RNews article and a quick glance the examples provided by James Wettenhall.

The following is a start, although we quickly run into issues that hopefully someone more knowledgeable about **tcltk** can resolve.

First we load the package.

```
> options(guiToolkit = NA)
> library(gWidgets)
> library(tcltk)

Loading Tcl/Tk interface ... done
```

Now we make subclass `guiWidgetsToolkit` so that we can dispatch on the toolkit.

```
> setClass("guiWidgetsToolkitTcltk", contains = "guiWidgetsToolkit",
+     prototype = prototype(new("guiWidgetsToolkit")))

[1] "guiWidgetsToolkitTcltk"

> guitoolkit = new("guiWidgetsToolkitTcltk")
```

Now we make a base class for the tcltk widgets created here.

```
> setClass("gWidgetTcltk")

[1] "gWidgetTcltk"

> setClass("guiWidgetORgWidgetTcltkORtcltk")

[1] "guiWidgetORgWidgetTcltkORtcltk"

> setIs("guiWidget", "guiWidgetORgWidgetTcltkORtcltk")
> setIs("gWidgetTcltk", "guiWidgetORgWidgetTcltkORtcltk")
```

Finally, we promote the `tkwin` class to an S4 class and add it to our virtual class. This would be done for all possible classes of `tcltk` objects.

```
> oldclasses = c("tkwin")
> for (i in oldclasses) {
+     setOldClass(i)
+     setIs(i, "guiWidgetORgWidgetTcltkORtcltk")
+ }
```

The gWidgetTcltk class is a virtual class, here are two subclasses. We create slots for the widget and the toolkit here, but perhaps should add others.

```
> setClass("gComponentTcltk", representation(widget = "guiWidgetORgWidgetTcltkORtcl
+     toolkit = "guiWidgetsToolkit"), contains = "gWidgetTcltk",
+     )

[1] "gComponentTcltk"

> setClass("gContainerTcltk", representation(widget = "guiWidgetORgWidgetTcltkORtcl
+     toolkit = "guiWidgetsToolkit"), contains = "gWidgetTcltk",
+     )

[1] "gContainerTcltk"
```

Now we define some necessary functions to implement gwindow() in the toolkit. This involves defining a class, make a constructor (.gwindow()) and defining some methods.

```
> setClass("gWindowTcltk", contains = "gContainerTcltk", prototype = prototype(new(

[1] "gWindowTcltk"
```

This implementation of the constructor should add a handler for the destroy event.

```
> setMethod(".gwindow", signature(toolkit = "guiWidgetsToolkitTcltk"),
+     function(toolkit, title = "Window", visible = TRUE, handler = NULL,
+         action = NULL, ...) {
+         win <- tktoplevel()
+         tktitle(win) <- title
+         obj = new("gWindowTcltk", widget = win, toolkit = toolkit)
+         return(obj)
+     })
```

```
[1] ".gwindow"
```

The `svalue()` method for `gwindow()` objects is used to retrieve and set the title of the window.

```
> setMethod(".svalue", signature(toolkit = "guiWidgetsToolkitTcltk",
+     obj = "gWindowTcltk"), function(obj, toolkit, index = NULL,
+     drop = NULL, ..) {
+     tktitle(obj@widget)
+ })
```

```
[1] ".svalue"
```

```
> setMethod(".svalue<-", signature(toolkit = "guiWidgetsToolkitTcltk",
+     obj = "gWindowTcltk"), function(obj, toolkit, index = NULL,
+     ..., value) {
+     tktitle(obj@widget) <- value
+     return(obj)
+ })
```

```
[1] ".svalue<-"
```

The `add()` method is used to add a widget to a container. This is where we run into problems with **tcltk** as the constructors there require a "parent" container at the time of construction. As such, we don't have both a container (`obj` below) and widget (`value`) needed when we add, rather we only specify how things are packed in.

```
> setMethod(".add", signature(toolkit = "guiWidgetsToolkitTcltk",
+     obj = "gWindowTcltk", value = "guiWidget"), function(obj,
+     toolkit, value, ...) {
+     tkpack(value@widget@widget)
+ })
```

```
[1] ".add"
```

The dispose method closes the window

```
> setMethod(".dispose", signature(toolkit = "guiWidgetsToolkitTcltk",
+     obj = "gWindowTcltk"), function(obj, toolkit, ...) {
+     tkdestroy(obj@widget)
+ })
```

```
[1] ".dispose"
```

Below we implement the basics of `glabel()`. No attempt is made to add a click handler to this or editing or markup. For now, just setting of text in a label.

First a class

```
> setClass("gLabelTcltk", contains = "gComponentTcltk", prototype = prototype(new("
```

```
[1] "gLabelTcltk"
```

Next the constructor

```
> setMethod(".glabel", signature(toolkit = "guiWidgetsToolkitTcltk"),
+     function(toolkit, text = "", markup = FALSE, editable = FALSE,
+         handler = NULL, action = NULL, container = NULL, ...) {
+         if (is.null(container)) {
+             cat("Can't have an NULL container with tcltk")
+         }
+         if (is(container, "guiWidget"))
+             container = container@widget
+         if (is(container, "gContainerTcltk"))
+             container = container@widget
+         label = tklabel(container, text = text)
+         obj = new("gLabelTcltk", widget = label, toolkit = toolkit)
+         tkpack(label)
+         return(obj)
+     })
```

```
[1] ".glabel"
```

The `svalue()` method returns the label text, to be honest I don't know enough about the **tcltk** package to write this, although to set the text is easy.

```
> setMethod(".svalue", signature(toolkit = "guiWidgetsToolkitTcltk",
+     obj = "gLabelTcltk"), function(obj, toolkit, index = NULL,
+     drop = NULL, ..) {
+     cat("How to retrieve label text\n")
+ })
```

```
[1] ".svalue"
```

```
> setReplaceMethod(".svalue", signature(toolkit = "guiWidgetsToolkitTcltk",
+     obj = "gLabelTcltk"), function(obj, toolkit, index = NULL,
+     ..., value) {
+     tkconfigure(obj@widget, text = value)
+     return(obj)
+ })

[1] ".svalue<-"
```

For the `gbutton()` implementation we show how to add a handler in addition to implementing the `svalue()` method.

```
> setClass("gButtonTcltk", contains = "gComponentTcltk", prototype = prototype(new(

[1] "gButtonTcltk"
```

As for the constructor we have:

```
> setMethod(".gbutton", signature(toolkit = "guiWidgetsToolkitTcltk"),
+     function(toolkit, text = "", handler = NULL, action = NULL,
+         container = NULL, ...) {
+         if (!is.null(container)) {
+             topwin = container@widget@widget
+         }
+         else {
+             topwin = gwindow(toolkit = toolkit)@widget
+         }
+         button = tkbutton(topwin, text = text)
+         obj = new("gButtonTcltk", widget = button, toolkit = toolkit)
+         tkpack(obj@widget)
+         if (!is.null(handler))
+             .addhandlerclicked(obj, toolkit, handler = handler)
+         return(obj)
+     })

[1] ".gbutton"
```

In dealing with the handler, we used the private method defined below, rather than `addhandlerclicked()` as that method is for objects of class `guiWidget`, and not `gWidgetTcltk`. This awkwardness can be avoided by defining a method `addhandlerclicked` for objects of class `gWidgetTcltk` within the toolkit. For instance,

```
> setMethod("addhandlerclicked", signature(obj = "gWidgetTcltk"),
+     function(obj, handler = NULL, action = NULL, ...) {
+         .addhandlerclicked(obj, obj@toolkit, handler, action,
+             ...)
+     })

[1] "addhandlerclicked"
```

Again, `svalue()` should retrieve the text, it shouldn't be hard, but I don't know how. Below is how to set the button text.

```
> setReplaceMethod(".svalue", signature(toolkit = "guiWidgetsToolkitTcltk",
+     obj = "gButtonTcltk"), function(obj, toolkit, index = NULL,
+     ..., value) {
+     tkconfigure(obj@widget, text = value)
+     return(obj)
+ })

[1] ".svalue<-"
```

This sets up a click handler for a button. The handler should have a first argument which is a list with the object and the action value passed in. This isn't done below, as it isn't clear to me how to do so with just the `tkconfigure()` function.

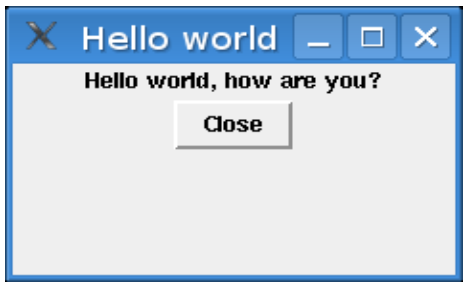

Figure 1: Hello world, how are you?

```
> setMethod(".addhandlerclicked", signature(toolkit = "guiWidgetsToolkitTcltk",
+     obj = "gButtonTcltk"), function(obj, toolkit, handler, action = NULL,
+     ...) {
+     tkconfigure(obj@widget, command = handler)
+ })
```

```
[1] ".addhandlerclicked"
```

Well, that will let us make the following simple dialog (Figure 1).

```
> win = gwindow("Hello world", toolkit = guitoolkit)
> label = glabel("Hello world, how are you?", container = win,
+     toolkit = guitoolkit)
> button = gbutton("Close", handler = function(h, ...) dispose(win),
+     container = win, toolkit = guitoolkit)
```

We have to specify a container each time and for these constructors the toolkit. If we had written a package this latter could be avoided using the `guiToolkit()` function.