Adding a toolkit to gWidgets

John Verzani, gWidgetsRGtk@gmail.com

January 25, 2012

Abstract:

[**This package is now out of date. The gWidgetstcltk package has been written. This is here in case someone wants to port a different toolkit (RwxWidgets say).**]

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 possibly look like using the **tcltk** package.

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, a constructor, 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.

The classes for the toolkits `RGtk2`, `tcltk`, `rJava`, and `SJava` are already defined by gWidgets. These are named `guiWidgetsToolkit` plus the package name.

As such, the basic structure of a gWidgets implementation is to set up some classes, and a set of methods for dispatch. [1]

2 An example

This shows some of what is necessary to write an implementation of gWidgets for the **tcltk** package. It only assumes as much about **tcltk** as was learned by browsing Peter Dalgaard's informative RNews article and a quick glance at the examples provided by James Wettenhall.

First we load the package.

```
## load toolkit
options("guiToolkit"=NA)
library(gWidgets)
library(tcltk)
```

The options setting ensures no toolkit for gWidgets gets loaded.

We note the subclass of `guiWidgetsToolkit`, `guiWidgetsToolkittcltk` is already defined in `gWidgets`.

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

```
## some classes
setClass("gWidgetTcltk")
## A virtual class to hold tcltk object or guiWidget or gWidgetTcltk
setClass("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")
}
```

[1] Although it likely wasn't necessary, at the time of first writing a package, the dispatch was to a "dot" file. This caused an extra level to the dispatch, that while unfortunate, does not seem to be worth rewriting to avoid.

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 such as an ID slot for storing a unique ID per widget.

```
### Make some base classes
setClass("gComponentTcltk",
representation(
widget="guiWidgetORgWidgetTcltkORtcltk",
toolkit="guiWidgetsToolkit"
),
contains="gWidgetTcltk",
)
setClass("gContainerTcltk",
representation(
widget="guiWidgetORgWidgetTcltkORtcltk",
toolkit="guiWidgetsToolkit"
),
contains="gWidgetTcltk",
)
```

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

```
## top level window
setClass("gWindowTcltk",
contains="gContainerTcltk",
prototype=prototype(new("gContainerTcltk"))
)
```

This implementation of the constructor should have a handler for the window 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)
  })
```

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)
})

setMethod(".svalue<-",
signature(toolkit="guiWidgetsToolkittcltk",obj="gWindowTcltk"),
function(obj, toolkit, index=NULL,..., value) {
  ## set the title
  tktitle(obj@widget) <- value
  return(obj)
})
```

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, ...) {
  ## how to add?
  tkpack(value@widget@widget)
})
```

[To avoid this, the **gWidgetstcltk** package requires a container be specified when a widget is constructed. This container stores the top-level container so that a **tcltk** widget can be constructed. The `add` method is then rarely used publicly, but is useful when writing the constructor. It's arguments for placement of the widget are specified during the construction of the widget.]

The `dispose` method closes the window

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

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

```
############
## label class
setClass("gLabelTcltk",
contains="gComponentTcltk",
prototype=prototype(new("gComponentTcltk"))
)
```

Next the constructor

```
  ## constructor
  setMethod(".glabel",
  signature(toolkit="guiWidgetsToolkittcltk"),
  function(toolkit,
  text= "", markup = FALSE, editable = FALSE, handler = NULL,
  action = NULL, container = NULL,
  ...
  ) {

    ## if container is non null, we can evaluate expression
    if(is.null(container)) {
      cat("Can't have an NULL container with tcltk")
    }
    ## find tk container
```

```
  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)

  ## pack into container
  tkpack(label)

  ## add callback
  ## no callbacks for labels
  return(obj)
})
```

The `svalue()` method returns the label text, it requires a little tcltk voodoo.

```
setMethod(".svalue",
signature(toolkit="guiWidgetsToolkittcltk",obj="gLabelTcltk"),
function(obj, toolkit, index=NULL, drop=NULL, ..) {
  as.character(tkcget(obj@widget,"-text"))
})

## svalue<-
setReplaceMethod(".svalue",
signature(toolkit="guiWidgetsToolkittcltk",obj="gLabelTcltk"),
function(obj, toolkit, index=NULL, ..., value) {
  ## set the text
  tkconfigure(obj@widget, text=value)
  return(obj)
})
```

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

```
### button class
setClass("gButtonTcltk",
contains="gComponentTcltk",
prototype=prototype(new("gComponentTcltk"))
)
```

As for the constructor we have:

```
setMethod(".gbutton",
signature(toolkit="guiWidgetsToolkittcltk"),
function(toolkit,
text="", border=TRUE, 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)

})
```

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. [2] For instance,

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

[2]This is a result of the way dispatch was designed. The toolkit information is stored in a slot separate from the widget provided by the toolkit in the gWidget object. Dispatch occurs on both the object and the toolkit. The method with these signatures is the "dot" one implemented in the toolkit package.

(The internal function, `.addhandlerclicked`, uses lower case letters for now.)

The handler could be written as follows

```
setMethod(".addhandlerclicked",
          signature(toolkit="guiWidgetsToolkittcltk",obj="gWidgettcltk"),
          function(obj, toolkit,
                   handler, action=NULL, ...) {
                     tkbind(getWidget(obj),<Button-1>,
                     function(...) {
                       h = list(ref=obj, obj=obj, action=action)
                       handler(h,...)
                     })
})
```

Again, `svalue()` should retrieve the text and `svalue<-()` should set the text. Again, a little tcltk voodoo is used.

```
setMethod(".svalue",
signature(toolkit="guiWidgetsToolkittcltk",obj="gButtonTcltk"),
function(obj, toolkit, index=NULL, drop=NULL, ...) {
  val = paste(as.character(tkcget(obj@widget,"-text")))
  return(val)
})

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

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

```
guitoolkit = new("guiWidgetsToolkittcltk")
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)
```

Figure 1: Hello world, how are you?