

Package ‘gWidgetstcltk’

April 3, 2012

Version 0.0-49

Title Toolkit implementation of gWidgets for tcltk package

Author John Verzani

Maintainer John Verzani <gwidgetsrgtk@gmail.com>

Depends R (>= 2.12.0), methods, gWidgets(>= 0.0.46), tcltk(>= 2.7.0),tcltk2, digest

Suggests

Description Port of the gWidgets API to the tcltk package. Requires TK 8.5 or greater for the tile (ttk)widgets <http://www.tcl.tk/software/tcltk/8.5.tml>. This is the default on Windows. Under linux, Tk must be installed. Under Mac OS X (10.5) there are two options: native Tk or X11. For the native one, Tk must be upgraded. See www.tcl.tk to download source. Under the Mac it compiles and installs cleanly. For X11, tcltk libraries can be downloaded to augment the R binary package. See <http://cran.r-project.org/bin/macosx/tools/>. The gdf function requires the add on Tk package TkTable (<http://tktable.sourceforge.net/>).

License GPL (>= 2)

URL <http://gwidgets.r-forge.r-project.org/>

LazyLoad yes

Repository CRAN

Date/Publication 2012-04-03 07:09:50

R topics documented:

gWidgetstcltk-package 2

Index 8

gWidgetstcltk-package *Toolkit implementation of gWidgets for tcltk*

Description

Port of gWidgets API to tcltk. The gWidgets API is an abstract, lightweight means to interact with GUI toolkits. In this case, the tcltk toolkit.

Details

This file documents differences between **gWidgetstcltk** and the **gWidgets** API, which is documented both in the man pages for that package and in its vignette. The **gWidgetstcltk** package is not as complete as **gWidgetsRGtk2**. This is due to limitations in the base libraries implementing tcl/tk. This package was designed to work with the limited version that comes with the standard Windows installation of R.

Notes on this implementation:

The primary difference in this interface, as opposed to that for **RGtk2**, is that each widget requires a container when being constructed. The container is given to the `container` argument. The value may be the logical `TRUE` indicating that a new window is constructed, or a container widget.

Other differences are that tcltk does not seem to have a markup language like Pango for GTK or HTML for JAVA, as such the `markup` argument for several widgets that is used to format text is not available. The `font<-` method can substitute.

Until version 8.5 of tk, the basic tcltk installation did not include several widgets that appear in other toolkits. For instance a grid widget, a notebook widget, etc. This package now requires tk 8.5 to work and R 2.12-0 or newer.

Containers:

`gwindow()`: The `width=` and `height=` arguments refer to the minimum window size, not the preferred default size. It is best to set `visible=FALSE` for the constructor and then when the GUI is layed out call `visible<-`. This will get the proper size for the window. Otherwise, the `update` method can be called to resize the window to accomodate the child widgets.

The methods `addHandlerDestroy` and `addHandlerUnrealize` can only add one handler, new ones overwrite the old. These handlers can also not be removed.

`ggroup()` has the `expand=` `fill=` and `anchor=` arguments. If `expand=TRUE` the widget is allocated as much space as possible. (The default is `expand=FALSE`, unless the option `"gw: tcltkDefaultExpand"` overrides this.) When a widget has `expand`, then the widget may stretch to fill the expanding space (in tcltk, all widgets that have `expand=TRUE` are allocated evenly any additional space). The `fill` may be `TRUE` (or both), `FALSE`, or `"x"` and `"y"`. The `anchor=` argument adjusts a widget left or right, up or down, within its space. Only one component works at a time for the `anchor` argument. In a horizontal box, only the `y` component can be used to move a value up or down. In a vertical box, only the `x` component can be used to move a value left or right. The default is `c(-1,0)` so

that horizontal layouts are towards the middle, and vertical layouts towards the left. (This can be overridden: `options("gw:tcltkDefaultAnchor"=c(0,0))`, say.)

The `use.scrollwindows` feature is now implemented. (but seems buggy)

The `addSpring` method only works if the parent container is set to expand.

`gframe()` The markup argument is ignored. Use `font<-` to give the title markup.

`gexpandgroup()` Works as expected, although sizing issues may frustrate.

`gdfnotebook()` Works with the addition of `ttknotebook`. The `add` method, which is used to add pages, is called when the notebook is given as a container during the construction of a widget. Hence, to add a page something like this is done:

```
nb <- gnotebook(cont=gwindow("Notebook example"))
gbutton("Page 1", cont=nb, label = "tab1")
glabel("Page 2", cont=nb, label = "tab2")
gedit("Page 3", cont=nb, label = "tab3")
```

`glayout()` has two additional arguments: `expand=TRUE` is like `expand=` for `ggroup()`, in that the attached widget expands to fill the possible space in the container. If this isn't given the `anchor=` argument can be used to adjust the location of the widget within the cell. A value of `c(-1,1)` is the lower left, `c(-1,-1)` the upper left (the default), `c(1,-1)` the lower right, and `c(1,1)` the upper right. The value 0 for either is also possible.

`gpanedgroup()` The constructor is called with no widgets. To add a widget to the paned group the paned group is passed as a container, as in

```
pg <- gpanedgroup(container=gwindow("example"), horizontal = FALSE)
b1 = gbutton("button 1", container=pg)
b2 = gbutton("button 2", container=pg)
```

The paned window can be adjusted manually or using the `svalue` method. The `svalue` method uses the current window size. If the widget is not realized, the method will not work as expected, so call this after showing the GUI. The `delete` method can be used to delete a widget. It may be added back with the `add` method.

The basic widgets or components: (These are also known as controls)

`gbutton()` mostly works. The button won't resize to take up all the possible space for a widget, even if `expand=TRUE` is given.

`gcalendar()` is a hack.

`gcheckbox()` works as advertised, `use.togglebutton` implemented.

`gcheckboxgroup()` works as advertised, except the `use.table` argument is ignored. One can now resize the list.

gcombobox() Works as expected, although no icons or tooltips are available.

gdf() is implemented if the user has installed the tktable package in Tcl. This is an additional download from tktable.sourceforge.net. Most of the code comes second hand from **tcltk2**'s dfedit function.

gedit(): The widget does not resize automatically. Set the width with width= at time of construction or with size<-. There is now type ahead support, although the pop-down menu only pops down, so don't use near the bottom of a screen ;) The hidden argument init_msg can be used to place an initial message for the event there is no text in the box.

gfilebrowse() works.

ggraphics() Not implemented. The **tkrplot** package could be used in some way, but this does not provide a fully embeddable graphics device. The **tkrplot** package provides a means to create interactive graphics with **tcltk**. This is not a device, so isn't directly supported. However, a ggroup object can be used as a parent container. Just call getToolkitWidget on the object first:

```
g <- ggroup(cont=gwindow())
l <- tkrplot(getToolkitWidget(g), function() hist(rnorm(100)))
add(g, l)
```

ghelp() Works as advertised. Uses a popup menu instead of a notebook, as gWidgetsRGtk2. Best to just use **helpr** though.

gimage() Only works with gif and pnm files, as the underlying tcltk widget only uses these by default.

glabel() No markup available. Use font<- instead.

gmenu() adds only to the top window, not any container. This is a tcltk limitation. Use a popupmenu instead.

Under Mac OS X, menus display in the top menu bar area, not in the parent window.

gtoolbar() A hack made from a ggroup object that packs in gbutton instances. The buttons take up alot of screen real estate, on the default Aqua them of OS X the buttons are rounded, so the toolbar looks odd, ...

gaction() is implemented for buttons, menubars and toolbars. The key.accel component is now implemented but one must pass in a parent argument (The binding is to the top-level window containing the parent).

gradio() has an extra argument coerce.with=, as otherwise it would treat everything as a character vector. It tries to guess when instantiated, if not explicitly given. One can now resize the number

of items to select from.

`gseparator()` works as expected but must be in a container with `expand=TRUE`.

`gslider()` now works with non-integer steps. If first argument `from` is a vector it will slide over those values after sorting. This uses a themed widget which might be buggy under some styles.

`gspinbutton()` Works as expected. The change handler responds to clicks of the arrows or to the return key in the text area. Unless one has a new Tk version, this is a non-themed widget and can look a bit odd.

`gstatusbar()` A hack. Just a `ggroup()` instance in disguise. By default it must have a `gwindow` instance for a parent container. If the hidden argument `not.toplevel=TRUE` is specified, a `ggroup` container may be used.

`gtable()` This is built on the underlying tree widget. It is not ideal, but avoids needing to have a separate library (eg. `Tktable`) installed. If the hidden argument `round` is passed to the constructor, this will be passed to the `format` function's `digits` argument when formatting numeric values for display.

Sizing is an issue. There may be a bug in the widget regarding horizontal scrolling (for Mac OS X anyways, where this is being developed), or more likely something is just coded wrong. There is some Tk code for "autoscrolling" that works (with an idiosyncrasy) so that the initial size of the widget is correct, but only when this size is set via arguments `width` and `height` passed to the constructor – not with the `size<-` method. This feature is not on by default, as when it is used any widgets on the right of the table are not shown in the initial window, and are only exposed by resizing the window. If you want to try it, pass in the hidden argument `do.autoscroll=TRUE`. However, the `size<-` method has another use. It can also take a list for value. This list has optional components `width`, `height`, `columnWidths` (to specify each column width individually), and `noRowsVisible` (to specify height by number of rows, not pixels).

`gtext()` The `size<-` method can be used to resize the widget. The initial default size is quite large. This method guesses at the conversion from pixels to characters (width) and lines of text (height) used by the underlying widget. The `svalue()` method returns all the text unless some text is selected by the mouse and `:index=TRUE` in which case the indices of the selected text are returned or `drop=TRUE` in which case only the selected text is returned.

`gtree()` Implemented using `ttktreeview`. It is slow however, so use on smaller trees only. Has same issues with scrollbars as `gtable`.

Compound components:

`gcommandline()` is implemented, but could definitely be improved.

`ghelpbrowser()` just calls `ghelp`

ggenericwidget() Some kinks need ironing out when the main variable is a formula.

gdfnotebook() Not implemented.

ggraphicsnotebook() No ggraphics so no notebook.

gvarbrowser() Uses a tree to show heirarchical structure of workspace. Does not poll to update workspace. It does reread workspace when Filter by: value is changed.

Dialogs: (These are modal, hence they have no methods (basically).)

gfile() works as advertised.

galert() works.

gmessage() works.

gconfirm() works.

ginput() works.

gbasicdialog() is implemented. It is a container. When the visible(obj, TRUE) command is issued, the container is shown and made modal.

```
dlg <- gbasicdialog("A modal dialog", handler=function(h,...) print("hi"))
l = glabel("some widget in the dialog", cont=dlg)
visible(dlg, set=TRUE)
```

Handlers:

Handlers were rewritten so that one can have more than one handler per signal. The blockHandler, unblockHandler and removeHandler methods are now working. Handler code different for those widgets which use an R5 backend and those which don't, but the end user shouldn't notice. (Well, if you do let me know!)

The addHandlerBlur method should be called when a widget loses focuses, but here is called whenever a widget loses focus *and* whenever the mouse leaves the widget. This can mean the handler is called twice. If you don't like that, you can add the callback through addHandler(obj, signal, handler) where signal is <FocusOut> or <Leave>.

adddroptarget(), adddropsource(), and adddropmotion work for tcltk widgets. The cursor changes to a funny looking cursor, but nothing resembling a drag and drop cursor. One was chosen from the standard cursors. Dragging from other applications is not supported.

Author(s)

John Verzani. Several code segments were inspired by the examples of Wettenhall and the Dalgaard article referenced below. The drag and drop code was modified from code at <http://wiki.tcl.tk/416>. Icons were "borrowed" from several places: the scigraphica project, KDE, and elsewhere.

Maintainer: John Verzani <gwidgetsrgtk@gmail.com>

References

Peter Dalgaard's RNews article on the tcltk package http://cran.r-project.org/doc/Rnews/Rnews_2001-3.pdf; Several examples on <http://bioinf.wehi.edu.au/~wettenhall/RTclTkExamples/>; The PERL documentation was useful: <http://search.cpan.org/~ni-s/Tk-804.027/>, although this is for the most part a translation of the TK documentation.

For a package somewhat similar in intent see Bowman, Crawford, Alexander, and Bowman's **rpanel** package: <http://www.jstatsoft.org/v17/i09/v17i09.pdf> or the **tkWidgets** package of Zhang www.bioconductor.org.

The **fgui** package provides a similar functionality as `ggenericwidget` in a standalone package.

Examples

```
## Not run:
##
options(guiToolkit="tcltk")
## select CRAN mirror example
setMirror = function(URL) {
  repos = getOption("repos")
  repos["CRAN"] = gsub("/$", "", URL)
  options(repos = repos)
}

win = gwindow("Select a CRAN mirror")
tbl = gtable(utils::getCRANmirrors(),
  container=win,
  chosencol=4,
  handler = function(h,...) {
    URL = svalue(h$obj)
    setMirror(URL)
    dispose(win)
  })
##

## End(Not run)
```

Index

*Topic **package**

gWidgetstcltk-package, [2](#)

gWidgetstcltk (gWidgetstcltk-package), [2](#)

gWidgetstcltk-package, [2](#)