gWidgetsWWW

John Verzani, `gWidgets@gmail.com`

August 7, 2009

Abstract:

The **gWidgets** package provides an API to abstract the interface for a few of the available GUI toolkits avaiilable through R. The **gWidgetsWWW** package provides an implementation of the **gWidgets** API for use with through the web. Using just R commands, interactive GUIs can be produced.

The current status of the project is still experimental. The package does not have much testing as of yet. As of version 0-0.8 the package works with internet explorer. Although some widgets (gcanvas, gsvg) are not working equally for all browsers.

1 Overview

To create a web GUI a means must be provided to callback to the web server when the user initiates an action and then the web server responds with commands to manipulate the user's page. The RApache package `http://biostat.mc.vanderbilt.edu/rapache/`, which embeds an R process within the Apache web server, is used so that callbacks from the browser to the web server can be processed through R. To return instructions to the page, javascript is used so that the entire page need not be reloaded, as javascript can manipulate elements on the page. The javascript code is simplified by using the extjs javascript libraries, available from `www.extjs.com`.

To make a GUI in **gWidgets** can be as easy as loading the **gWidgets** package then calling

```
w <- gwindow("simple GUI with one button", visible=FALSE)
g <- ggroup(cont = w)
b <- gbutton("click me", cont = g, handler = function(h,...) {
```

```
    gmessage("hello world", parent = b)
})
visible(w) <- TRUE
```

To run the GUI, two ways are possible. This code can be embedded in (or sourced from) a **brew** template and run through the combination of **brew** and **RApache**. Alternatively, one can configure **gWidgetsWWW** to process specific URLs through this template and encode the file name in the URL. With this approach, adding a web page is a easy as adding a file to a directory and pointing a browser at it. The main webpage for **gWidgetsWWW** http://www.math.csi.cuny.edu/gWidgetsWWW contains examples of some basic GUIs. These are included in the Examples subdirectory of the package once installed.

2   Top level windows

Web GUIs are different than desktop GUIs. Not only are they slower, as they have lag time between the GUI and the server, there can only be one top-level windows. The gwindow call above makes such a top-level window (a web page). Subwindows are possible, but all other gwindow instances should use the parent argument to specify an object that acts as the parent of a subwindow. (an animation will appear from there, say.)

3   The containers

The **gWidgetsWWW** package has all the following containers:the top-level container gwindow, subwindows also constructed through gwindow (use a parent object); the box containersggroup, gframe and gexpandgroup; the tabular layout container glayout; the notebook container gnotebook, but no gpanedgroup.

To make a component appear in response to some action – such as happens with gexpandgroup, one can add it to a box container dynamically. Or one can put it in a ggroup instance and toggle that containers visibility with the visible method, in a manner identical to how gexpandgroup is used.

4   The widgets

Most – but not all – of the standard widgets work as expected. This includes labels (glabel), buttons (gbutton), radio buttons (gradio), checkboxes (gcheckbox,

gcheckboxgroup), comboboxes (`gcombobox`), sliders (`gslider`), spinboxes (well kind of) (`gspinbutton`), single-line edit boxes (`gedit`), multi-line text areas (`gtext`), dataframe viewers (`gtable`) and editors (`gdf`), images (`gimage` – the image is a url), menu bars (`gmenu` – but not toolbars), statusbars (`gstatusbar`).

Some of the dialogs work including `gcalendar`, `galert`, `gmessage`. But `gconfirm`, `ginput` and `gfile` are not working.

The widgets `galert`, `ghtml` and `gaction` are all implemented.

No attempt has been made to include the compound widgets `gvarbrowser`, `ggenericwidget`, `gdfnotebook`, `gcommandline` and `ggraphicsnotebook`.

4.1 graphics

There is no plot device avaiable. Rather, one uses the **Cairo** device driver to create graphic files which are then shown using `gimage`. The function `getStaticTempfile` should be used to produce a file, as this file will sit in a directory that can be accessed through a URL. This URL is returned by `convertStaticFileToUrl`. The **Cairo** package is used, as it does not depend on X11, so works in server installations as well. An example is provided with the package.

Two package-specific widgets, `gcanvas` and `gsvg`, can be used for displaying non-interactive graphics files through the **canvas** device or the `RSVGTipsDevice` (the `SVGAnnotation` device should work as well). There are used similarly: You create the widget, you create a file. The graphics device writes to the file (similar to the `png` device driver, say). This file can be assigned to the widget at construction time, or later through the `svalue` method. For `gsvg` the file must be accessible as a URL, so the `getStaticTempfile` function should be used. The **canvas** device uses a newer HTML entity, `canvas`, which is not supported on all browsers. The `gsvg` package uses a SVG (scalable vector graphics) format. This format again has some issues with browsers, but seemingly fewere. The **RSVTTipsDevice** device has simple features for adding tooltips and URLs to mouse events. The **SVGAnnotation** package allows this an much more.

4.2 Quirks

A number of little quirks are present, that are not present with other **gWidgets** implementations:

1. The top level window is not made visible at first. (A good idea in any case, but not the default for `gwindow`. To create a window the `visible` method, as

in `visible(w) = TRUE`, is used. (When this is issue the javascript to make the page is generated.)

2. The handlers are run in an environment that does not remember the loading of any packages beyond the base packages and **gWidgetsWWW**. So in each handler, any external packages must be loaded.

3. The use of the options `quietly=TRUE` *and* `warn=FALSE` should be used with `require` when loading in external packages. Otherwise, these messages will be interpreted by the web server and an error will usually occur.

4. Debugging can be tough as the R session is not readily available. The error messages in the browser are useful

4.3   Data persistence

AJAX technologies are use to prevent a page load every time a request is made of the server, but Each time a page is loaded a new R session is loaded. Any variables stored in a previous are forgotten. To keep data persistent across pages, one can load and write data to a file or a data base.

4.4   Comboboxes

The `gcombobox` example shows how comboboxes can show a vector of items, or a data frame with a column indicating an icon, or even a third icon with a tooltip. As well, the `gedit` widget does not have a type-ahead feature, but the combobox can be used for this purpose.

The following will set this up.

```
> cb <- gcombobox(state.name, editable=TRUE, cont = w)
> cb$..hideTrigger <- TRUE ## set property before being rendered
```

This package is implemented independently of the **gWidgetsWWW** package, and so there may be some unintended inconsistencies in the arguments. The package uses the **proto** package for object-oriented support, not S3 or S4 classes. There are some advantages to this, and some drawbacks. One advantage is the user can modify objects or call their internal methods.

4.5 ggooglemaps

The `ggooglemaps` widget provides access to a sliver of the google maps API. This sliver could be enlarged quite easily if desired. Using this requires the web server to be registred with google.

5 Installation

Installation of the package requires several steps:

1. Install the **gWidgetsWWW** package from CRAN

2. Install the RApache module. The RApache web page has installation information. This APache module is available for UNIX and MAC OS X. Windows users can use the author-provided vmware appliance (`www.vmware.com`).

3. Configure the RApache module. A template is included with the variables that are set in this module. There are several little steps

   - Specifying the URL for the extjs javascript libraries (below)
   - Specifying the script where the session file lives (the script is in the package)
   - Specifying where a web-server writable directory is by absolute path and URL for holding the static files created by **gWidgetsWWW**. (This is needed for creating graphics files.)
   - Specifying how files are turned into web pages. The easiest way is the use the gWidgetsWWWrun script to specify directories where the files are held and to use the appropriate URL. (In can be as easy as specifying `run` as the script name, then `run/file` will find the file `file.R` in one of the directories and run that.

     The other option – and both can be used – is to use **brew**, as is described in the RApache manual. This requires using a brew template. One is provided in the Examples directory.

4. Install the ext javascript libraries so that they have a URL specified above. These libraries are free to use for open-source use (`http://www.extjs.com/license`) and can be found at `www.extjs.com`. Version 3.0 or higher is needed.

5. Copy the button images from the **gWidgetsWWW** package to an images URL.

6 Security

Security – is a big deal. Web servers can be hacked, and if hacked the hacker has full access to the server. This can be scary. Although it is not believed that RApache is any less secure than other Apache modules, you can protect yourself by running the entire setup within a virtual machine. There is easy to install, reasonably priced (or free) commercial software from VMWare (`www.vmware.com`). For open-source fans, the VirtualBox project (`www.virtualbox.org`) also has software. One can install this, then run the author provided appliance. Or one can install the virtual software, install a host OS (ubuntu linux say), then install Apache and R then RApache etc. It actually isn't so hard to do.

The call from the web server back into RApache can also be source of insecurity. The **gWidgetsWWW** package allows only a limited number of calls back from a web page, which should in theory be secure. But if the script is not secure, there is nothing theh package can do. Scripts must *never* trust that data sent from the web page to the server is safe. It should be coerced into any desired format, and never evaluated. Using `eval` say allows any one to run R commands on the server which given the power of R means they have full control of the server.