

BUILDING A GUI IN WINDOWS WITH GWIDGETSRGTK2

Giles Heywood, Amber Alpha
Presented at London R User Group

Contents

- gWidgets as a uniform interface to lower-level toolkits: tcltk, RGtk2, ...
- gWidgets packages, toolkits, graphics, containers, widgets, and their methods
- Beyond the basic widget: layout, grouping, addressing, refreshing
- Some minor niggles
- A full-scale application

gWidgets

- Toolkits

- Gtk2, the most complete, requires installation of GTK+ runtime library
- TclTk, not as polished, but tcltk package is distributed with R
- rJava, may soon be deprecated
- WWW, for use with webpages

| Package | Depends/Imports/Suggests | Author | Maintainer | Licence |
|-----------------|---|---------------------------------|--------------|---------|
| gWidgetsRGtk2 | methods, gWidgets, RGtk2, cairoDevice | John Verzani & Michael Lawrence | John Verzani | GPL |
| gWidgetstcltk | methods, gWidgets, tcltk, digest | John Verzani | John Verzani | GPL |
| gWidgetsWWW | methods, proto, filehash, digest, rjson | John Verzani | John Verzani | GPL |
| gWidgetsrJava | methods, gWidgets, rJava | John Verzani | John Verzani | GPL |
| ? (forthcoming) | RwxWidgets | | | |

- Graphics: the dependencies

- gWidgets < gWidgetsRGtk2 < RGtk2, cairoDevice < GTK+, Cairo
- Cairo and GTK+ are licenced under Lesser GPL

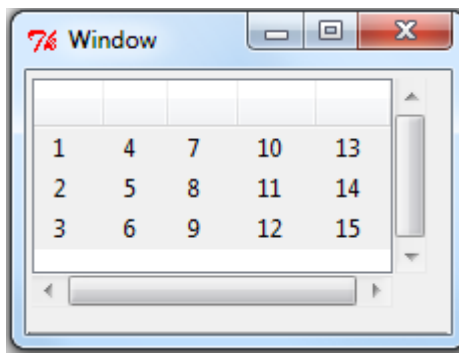
| Package | System requirements | Author | Maintainer | Licence |
|---------|---------------------------|---------------------------------------|------------------|---------|
| RGtk2 | Cairo, ATK , Pango , GTK+ | Michael Lawrence & Duncan Temple Lang | Michael Lawrence | GPL |

Toolkit aesthetics

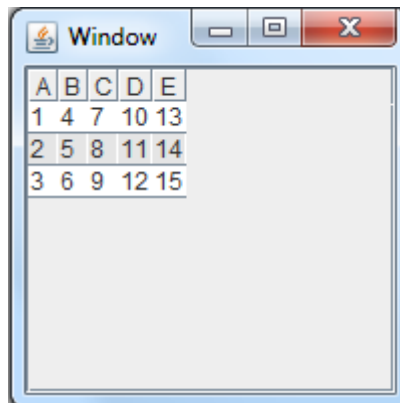
- Example of a simple widget three ways

```
require("RGtk2")
require("gWidgets")
require("rJava")
require("cairoDevice")
mytoolkits <- c("tcltk","rJava","RGtk2")
for(i in 1:3) {
  require(package=paste("gWidgets",mytoolkits[i],sep=""),char=TRUE)
  gtable(
    matrix(1:50,5,10),
    toolkit=guiToolkit(mytoolkits[i]),
    container=gwindow(toolkit=guiToolkit(mytoolkits[i])))
}
```

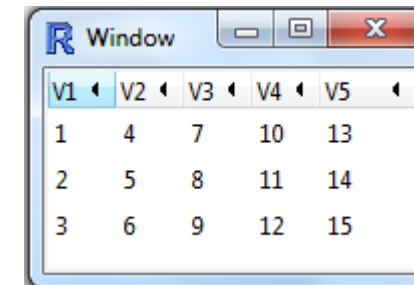
- Tcltk



- java



- GTK



Selected containers and widgets

• Containers

- `gwindow`: top level container
- `ggroup`: vanilla sub-container
- `gframe`: group with outline plus label
- `gpandedgroup`: 2-paned window with adjustable divider
- `glayout`: grid for organising a tabular layout
- `gnotebook`: tabbed notebook for organising multiple pages

```
panedgroup(widget1=gframe("first"), widget2=gframe("second"), container=gwindow())
```

• Selected non-trivial widgets

- `gtoolbar`: one of the neater ways of organising buttons, using icons
- `gtree`: for heirarchical data
- `gtable`: table, sortable but not editable
- `gdf`: table, editable and subsettable but not sortable

• Icons

```
mygrid <- glayout(container=gwindow(width=300,height=400),use.scroll=TRUE)
for(i in 1:length(getStockIcons()))
  mygrid[1+i%%10,i%%10+1] <-
    gimage(getStockIcons()[i],dirname="stock",size="button")
```

Selected handlers and methods

- **Handlers**

- Can be declared with the widget for its default action

```
gbutton("press me",
       container=gwindow(),
       expand=FALSE,
       handler=function(...) {ggraphics(cont=gwindow()); Sys.sleep(1); barplot(1:3)})
```

- Or added later with the `addHandler()` methods

| | arguments |
|------------------------------------|------------------------------------|
| <code>addHandlerchanged</code> | <code>obj, handler, action</code> |
| <code>addHandlerclicked</code> | <code>obj, handler, action</code> |
| <code>addHandlerdoubleclick</code> | <code>obj, handler, action</code> |
| <code>add3rdmousepopupmenu</code> | <code>obj, menulist, action</code> |
| <code>removeHandler</code> | <code>obj, ID</code> |

- **Some frequently-used methods**

| | arguments |
|------------------------------------|------------------------------------|
| <code>add</code> | <code>object, value, expand</code> |
| <code>delete</code> | <code>object, widget</code> |
| <code>svalue, svalue<-</code> | <code>obj, index</code> |
| <code>enabled, enabled<-</code> | <code>obj, value</code> |
| <code>visible, visible<-</code> | <code>obj, value</code> |

Basic layout principles

- Selecting the layout with `add(expand)`
- Can be quite verbose
 - #Fill the pane

```
gw0 <- ggroup(horizontal=TRUE, container=gwindow())
gw1 <- gframe("gw1",horizontal=FALSE)
add(gw1,gbutton("1a"),expand=TRUE)
add(gw1,gbutton("1b"),expand=TRUE)
add(gw0,gw1,expand=TRUE)
```

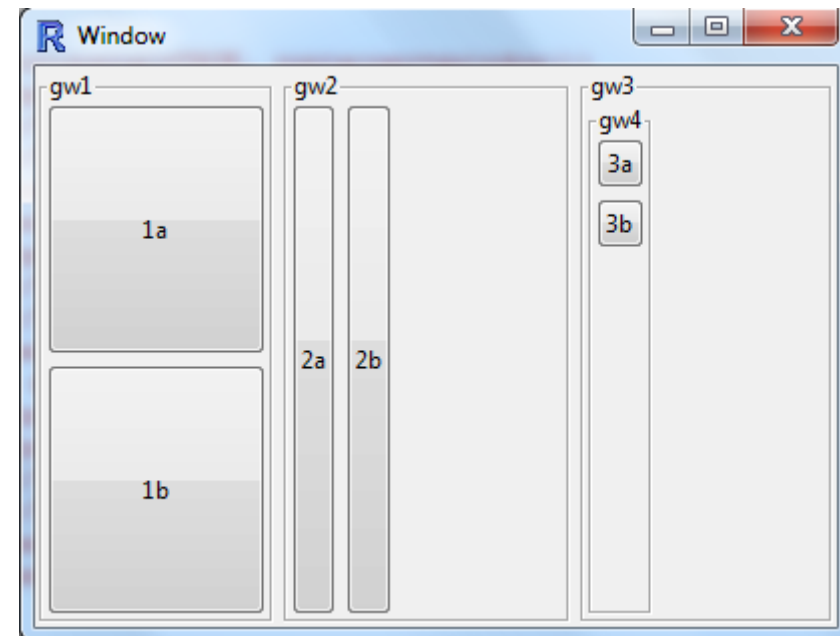
- #Vertical expansion only

```
gw2 <- gframe("gw2",horizontal=TRUE)
add(gw2,gbutton("2a"),expand=FALSE)
add(gw2,gbutton("2b"),expand=FALSE)
add(gw0,gw2,expand=TRUE)
```

- #No expansion – minimal dimensions

```
gw3 <- gframe("gw3",horizontal=TRUE)
gw4 <- gframe("gw4",horizontal=FALSE)
add(gw4,gbutton("3a"),expand=FALSE)
add(gw4,gbutton("3b"),expand=FALSE)
add(gw3,gw4,expand=FALSE)
add(gw0,gw3,expand=TRUE)
```

- see also: `addSpring()`



Organising an application

- For larger applications, the following needs arise:
 - Structure the layout into pages and subpages (modules)
 - Access selected values across modules using `select()`
 - Group widgets into modules, and refresh widgets or 'rebuild' modules
- Other requirements
 - Disabling modules during longer computations
 - Confirmation dialogue
- One solution
 - Addressing widgets
 - Store all widgets in a recursive list as a global object
 - Optionally store in a namespace to avoid use of the global assignment <<-
 - Addressing containers
 - Construct each widget module within a pair of containers:
 - Outer permanent container
 - Inner temporary container
 - Store the containers in the same global object
 - To build/rebuild the module, discard the temporary container and rebuild it

Organising a complex container into modules (1)

- Add toolbar, handlers

```
`nestof2` <- function(horizontal=FALSE,...) {
list(perm=gframe("perm",horizontal=FALSE),temp=gframe("temp",horizontal=horizontal,...
))}
```

```
`rus1` <- function(toolbar=FALSE) { #nested notebooks
  rus <- list(universe=NULL,lib=NULL,nb=gnotebook())
  rus$universe <- list(security=nestof2(),other=nestof2(),nb=gnotebook())
  rus$universe$lib <- nestof2(horizontal=TRUE)
  add(rus$nb, rus$universe$nb, label="universe")
    add(rus$universe$nb, rus$universe$security$perm, label="security")
    add(rus$universe$nb, rus$universe$other$perm, label="other")
  add(rus$nb, rus$universe$lib$perm, label="library")
  if(toolbar) {rus2(container=rus$universe$security$perm)}
  win <- gwindow(parent=c(100,250),width=400*1.618,height=400)
  add(win, rus$nb)
  svalue(rus$nb) <- 1
  svalue(rus$universe$nb) <- 1
}
```

```
rus1(toolbar=FALSE)
```

Organising a complex container into modules (2)

- Add toolbar, handlers

```
`rus2` <- function(container) { #add toolbar
  tbl <- list(
    refresh=gaction(icon="refresh",label="refresh",handler = rus3),
    image=gaction(icon="plot",label="image",handler = rus4)
  )
  add(container,gtoolbar(tbl))
  container
}
`rus3` <- function(...) { #add button
  delete(rus$universe$security$perm,rus$universe$security$temp)
  rus$universe$security$temp <<- gframe("temp")
  Sys.sleep(1)
  add(rus$universe$security$temp,gbutton("rus3"))
  add(rus$universe$security$perm,rus$universe$security$temp)
}
`rus4` <- function(...) { #add graphics device with optional delay
  mydelay <-
  ifelse(is.null(rus$universe$security$delay),1,as.numeric(svalue(rus$universe$security$delay)))
  delete(rus$universe$security$perm,rus$universe$security$temp)
  rus$universe$security$temp <<- gframe("temp")
  ggraphics(cont=rus$universe$security$temp)
  add(rus$universe$security$perm,rus$universe$security$temp)
  Sys.sleep(mydelay)
  barplot(1:3)
}

rus1(toolbar=TRUE)
```

Instance of lower-level bits you may need to do

- The confirmation dialog

```
`confirmDialog` = function(  
    message="do X?",  
    todo=print("X")  
)  
  
{  
    delayedAssign('promise', todo)  
    mywin = gwindow("Confirm", width=230, height=80, visible=FALSE)  
    group = ggroup(container = mywin)  
    gimage("dialog-question", dirname="stock", size="dialog", container=group)  
    inner.group = ggroup(horizontal=FALSE, container = group)  
    glabel(message, container=inner.group, expand=TRUE)  
    button.group = ggroup(container = inner.group)  
    addSpring(button.group)  
    gbutton("ok", handler=function(h,...){dispose(mywin);done<-promise},  
    container=button.group)  
    gbutton("cancel", handler =  
    function(h,...){dispose(mywin)}, container=button.group)  
    visible(mywin) <- TRUE  
}  
confirmDialog()
```

Performance and other issues

- Elapsed time and cpu time to add a (largeish) table or dataframe may be longer than anticipated

```
`rus5` <- function(..., type=c("tab", "dfr"), n=100) {      #delays for large n
  type <- match.arg(type)
  delete(rus$universe$security$perm, rus$universe$security$temp)
  tab <- matrix(1:500, 5, n)
  if(type=="tab") {widget <- gtable(tab)} else {widget <- gdf(tab)}
  rus$universe$security$temp <<- gframe("temp")
  add(rus$universe$security$temp, widget, expand=TRUE)
  add(rus$universe$security$perm, rus$universe$security$temp, expand=TRUE)
}
rus1(toolbar=TRUE)
rus5(n=300)
```

- Occasional issues with ggraphics – see earlier example requiring `Sys.sleep()`
- Occasional unexpected behaviour with alignment
- These are not major issues

Full-scale example

- Investment example: global equity market neutral backtest application
- R + MySQL + gWidgetsRGtk2 + many more
- Containers:
 - Basic: Window, Group, Frame
 - Notebook, Layout, Panedgroup
- Widgets
 - Basic: Text, Label
 - Toolbar, Button, Spinbutton, Checkbox, Radiobutton, Tree, Table, Dataframe
- In practice, gWidgetsRGtk2 provides a robust user interface - with thanks to John Verzani and many others
- Resources

GTK: www.gtk.org

Cairo: cairographics.org

gWidgets: wiener.math.csi.cuny.edu/pmg

