



---

*Proceedings of the 3rd International Workshop  
on Distributed Statistical Computing (DSC 2003)  
March 20–22, Vienna, Austria ISSN 1609-395X  
Kurt Hornik, Friedrich Leisch & Achim Zeileis (eds.)  
<http://www.ci.tuwien.ac.at/Conferences/DSC-2003/>*

---

# Putting RGtk to Work

James Robison-Cox

## Abstract

The RGtk package built by Duncan Temple Lang provides one way to build statistical applications in R which can be controlled by the user through a graphical user interface. RGtk is an interface to the GTK+ library of the GNU Project. In this paper an example GUI is built to illustrate usage of the RGtk package. The basic concepts of widgets and callbacks are demonstrated so that those familiar with programming in R will be able to build graphical applications.

## 1 Motivation

Graphical interfaces to statistical applications are useful for many purposes, including avoidance of the command line interface for repetitive analyses by non-statisticians, and for development of pedagogical tools. In this paper, the motivation for development will be a particular teaching tool. Any of the computer simulation modules described in Mills (2002) would be appropriate for a demonstration. I have chosen a regression demonstration which goes beyond the statistical capabilities of a spreadsheet, using a resistant regression routine and a smoothing routine. The R statistical program (Ihaka and Gentleman (1996)) will be used for the underlying computation. The regression demo is intended as an introduction to the concept of fitting a straight line to data points. The objective is to help students visualize how changes in a single point effect the least squares line. The intended audience for this demonstration is a group of neophytes in statistics; people who would be uncomfortable with a command-line interface. In fact, even if programming skill could be assumed, the ideal pedagogical technique would be graphical rather than command-line oriented so that the student's attention would be on the data and results rather than on the code used to invoke the demonstration.

In switching from the command line to a graphical interface, we are changing the types of events to which the program must respond. The R engine expects command line inputs, and, if so instructed, can also process mouse clicks in a plotting region with the `locator` function. In contrast, the GUI approach requires detection of any

of many possible user inputs, either by keystrokes in a text window or by mouse actions. These inputs will be translated into calls to R, and output from such calls will be used to update the display.

A variety of methods are available for producing graphical interfaces to R programs. Historically, [Dalgaard \(2001\)](#) introduced the `tcltk` package as the first interface to a graphical toolkit. Recently there has been a flurry of interest in GUI's for R in the Fall of 2002 on the R-Help mailing list (<http://www.stat.math.ethz.ch/pipermail/r-help/>) continued on ([http://www.sciviews.org/\\_rgui/](http://www.sciviews.org/_rgui/)) but no agreement on a favorite graphical toolkit. Many other graphical toolkits are being used to develop GUI's with R, for instance, `ObveRsive` (<http://obversive.sourceforge.net/>), a recent addition, uses the `FOX` graphical toolkit. Without claiming that `GTK` (from `GIMP ToolKit`, <http://www.gtk.org/>) is the best graphical toolkit for use with R, I would note that it is GPL, is available for many platforms including GNU/Linux, Unix, MS Windows, and MacOSX. The `RGtk` interface to `GTK` allows an R programmer to utilize the `GTK` library through R commands without reverting to programming in another language. Like most toolkits, `GTK` contains a full set of graphical tools, but the complexity comes at a price in terms of learning curve. The remainder of this paper is a brief tutorial describing how the regression demo can be built in `RGtk`. Initially a demo will be built with a minimal set of tools, then more widget will be introduced, and finally another library will be utilized to improve and extend the demo.

## 2 GTK

### 2.1 Widgets

The basic building blocks to be combined into a GUI are called widgets. Those of particular interest to the R programmer include menus, buttons, sliders, and text boxes for user input; text boxes and plotting areas for user output. A look ahead at [Figure 1](#) may be useful to see some of the widgets, including text boxes, which will can be used. Widgets are typically created, given the desired attributes, and then packed into windows. With `RGtk`, control over widgets utilizes the class attributes of objects to determine which `GTK` function to call. For example, in the code below, a simple text widget is created, and text is inserted into the widget.

```
> textx <- gtkText()      ## Creates a widget by calling S_gtk_text_new
> attributes(textx)      ## Check its attributes
$class
[1] "GtkText"      "GtkWidget"   "GtkObject"
> textx$insert(chars=xtxt, length= nchar(xtxt))      ## or
> gtkTextInsert(textx, chars=xtxt, length= nchar(x))
      ## both call the C routine S_gtk_text_insert
```

The last two lines perform the same action because `textx$insert` is interpreted by utilizing the class attribute of `textx`, `GtkText`, and is translated into a call to `gtkTextInsert`. Similarly one could create an object of class `GtkMenu` called `menu1` and use `menu1$insert()` to invoke the `gtkMenuInsert` function, a binding to `S_gtk_menu_insert`. The code which follows uses the shortened forms wherever possible.

The process of developing a graphical application involves:

1. Creating appropriate widgets,
2. Displaying the widgets,
3. Capturing user-initiated events, and
4. Updating output based on user inputs.

These four steps will be illustrated as the regression demo is built.

## 2.2 Widget creation

One starts by creating a window for display. It's easiest to display the window after all its internal constituents have been built, so the `show` argument is initially set to `FALSE`.

```
wRegDemo <- gtkWindow(show=FALSE)    ## Create Display Window
wRegDemo$SetTitle("Regression Demo")  ## Give it a title
```

Within the window we want several components. These will be “packed” into boxes. Boxes can be either vertical or horizontal stacks of widgets. We will use one of each.

```
box1 <- gtkVBox()    ## Create a Vertical box
wRegDemo$Add(box1)  ## Add it to the display window
box2 <- gtkHBox()    ## Create a Horizontal box
```

To keep the example simple, the  $x$  and  $y$  values for the regression will initially be displayed in text boxes which the user can edit. We will use separate text boxes for each variable.

```
x <- sort(round(rnorm(10, 100,10))) ## create some data (or use your own)
makeTextbox <- function(data){
  # a function to create and load an editable text box.
  box <- gtkText()
  box$SetEditable(TRUE)    ## Make it user-editable
  btxt <- paste(c(deparse(substitute(data)), as.character(data)), collapse="\n")
  box$Insert(chars=btxt, length= nchar(btxt))
  ## Insert the values into the text box
  return(box)
}
textx <- makeTextbox(x)    ## Create a text box for "x"
```

The response ( $y$ ) values are generated from the  $x$ 's and inserted as character values into their own text box.

```
y <- round(x + rnorm(10,0,4)) ## or use interesting data
texty <- makeTextbox(y)    ## Create a text box for "y"
```

Finally, we need a drawing region, so we will create a drawing area widget. In order to use a `GtkDrawingArea` widget as an R graphics device, we also need the `gtkDevice` package (Drake, et al. 2003).

```
require(gtkDevice)
drawArea <- gtkDrawingArea() ## Create the widget
drawArea$SetUsiZe(300,300)   ## specify the size
asGtkDevice(drawArea)       ## set as graphics device for R
```

### 2.3 Arranging and displaying widgets

For this simple example, the text and drawing area widgets will be displayed side-by-side in the horizontal box, `box2`. The `PackStart` function places them into `box2` from left to right (`PackEnd` would place them from right to left).

```
box2$PackStart(textx)
box2$PackStart(texty)
box2$PackStart(drawArea)
```

The horizontal box is ready to be placed into the vertical box, but first a label will be added to give the user minimal instructions.

```
label1 <- gtkLabel( "Change numerical values to move points.")
box1$PackStart(label1)
box1$PackStart(box2)
```

The only visual missing from our simple demo is the plot of the points. Since this will be repeated when points are changed, it needs to be a function. The original data were created above, but the user will be allowed to modify the data, in which case the plot is redrawn.

```
redraw <- function(x,y){
  ## function to plot points and draw regression line
  plot(y ~ x, xlab = "x",ylab="y", pch = 16, col=4)
  abline(lm(y~x))
}
redraw(x,y)
```

Finally we change the attribute of the display window to make everything appear, as shown in Figure 1.

```
wRegDemo$ShowAll()
```

### 2.4 Capturing user interaction and updating

The next task is to build callbacks which will register user inputs and update the displayed output. In this case, the user needs to be able to change the values of  $x$  and  $y$  coordinates. When such changes occur, the demo should redraw the points and the regression line. Registration of a callback requires specification of the widget in which an event is to be detected, the type of event to capture, and the function to be invoked. For the demo, callbacks are needed for each of the two text windows. When the user inserts text, the data will be updated and the plot redrawn (no action needed for text deletions).

```
regDF <- data.frame(x = x, y = y)
callbk <- function(varName,wdgt,...){
  tmp <- wdgt$GetChars(start=0, end=wdgt$GetLength())
  regDF[[varName]] <<- as.numeric(unlist(strsplit(tmp,"\n"))[-1])
  redraw(regDF$x, regDF$y) ## Uses R's scoping to find regDF
}
textx$AddCallback("insert-text", callbk, "x")
texty$AddCallback("insert-text", callbk, "y")
```

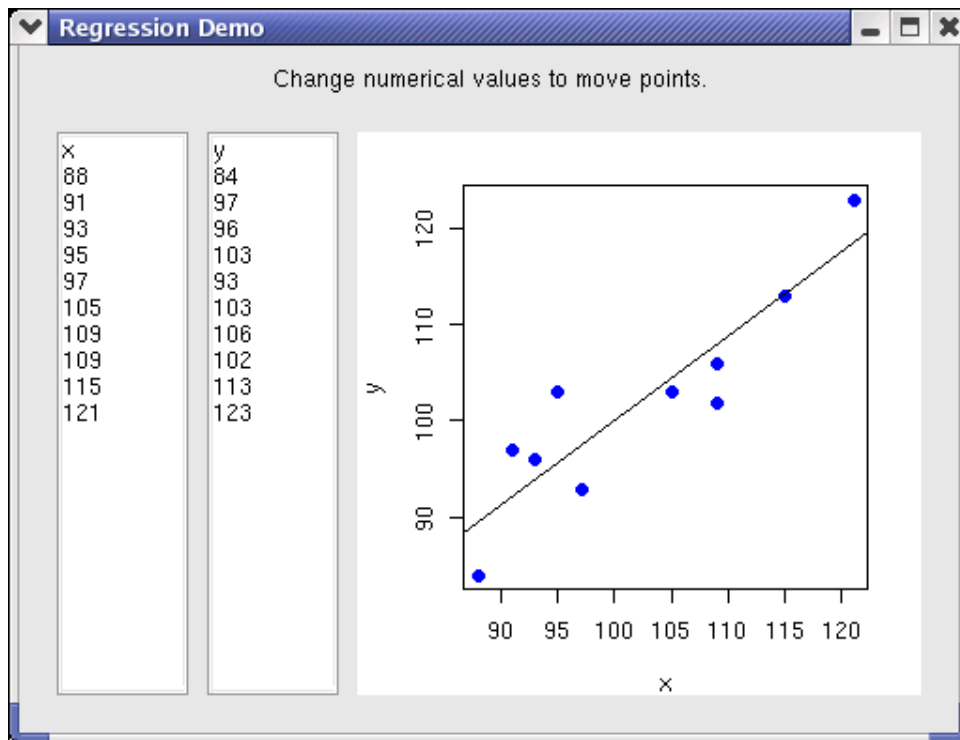


Figure 1: Screen-shot of the Simple Regression Demonstration

The above implementation is simple in that it uses only a few widgets, but clumsy in that if a user is inserting a number which is more than a single character, the callback is activated for each digit, rather than at the end of the inputs. The use of the “<<-” assignment operator was needed to change values in the global environment, but is considered a dangerous programming practice. Use of an “Update” button which the user would click after updating data would remove these problems. A complete code listing for the improved version with an update button is given. Frames around the data text boxes have been added to separate labels from data. The improved regression demo is illustrated in Figure 2 (with some extra features we’ll soon add).

```
wRegDemo <- gtkWindow(show=FALSE)
wRegDemo$SetTitle("Regression Demo")
box1 <- gtkVBox()
wRegDemo$Add(box1)
makeTextbox <- function(data){
  # a function to create and load an editable text box.
  box <- gtkText()
  box$SetEditable(TRUE)
  btxt <- paste(as.character(data), collapse="\n")
  box$Insert(chars=btxt, length= nchar(btxt))
  return(box)
}
```

```

x <- sort(round(rnorm(10, 100,10))) ## Generate data (or use your own).
frame.x <- gtkFrame("x")          ## Create a frame for "x".
textx <- makeTextbox(x)           ## Create a text box for "x"
frame.x$Add(textx)                ## place text inside the X frame
y <- round(x + rnorm(10,0,4))      ## or use interesting data
frame.y <- gtkFrame("y")          ## Create a frame for "y".
texty <- makeTextbox(y)           ## Create a text box for "y"
frame.y$Add(texty)                ## placed inside the Y frame
require(gtkDevice)

drawArea <- gtkDrawingArea()      ## Create the widget.
drawArea$SetUsiZe(300,300)        ## Specify the size.
asGtkDevice(drawArea)             ## Set as graphics device for R.
box2 <- gtkHBox()                 ## horizontal box filled with:
box2$PackStart(frame.x)           ## x data,
box2$PackStart(frame.y)           ## y data, and
box2$PackStart(drawArea)          ## plotting area
label1 <- gtkLabel("Change numerical values to move points.")
box1$PackStart(label1)
update1 <- gtkButtonNewWithLabel("Then click to update the plot.")
box1$PackStart(update1)
box1$PackStart(box2)
wRegDemo$ShowAll()
getNewVals <- function(){
  ## function to read data
  x <- as.numeric(unlist(strsplit(textx$GetChars(start=0,
                                   end=textx$GetLength()), "\n")))
  y <- as.numeric(unlist(strsplit(texty$GetChars(start=0,
                                   end=texty$GetLength()), "\n")))
  return(data.frame(x=x,y=y))
}
redraw <- function(x,y){
  ## function to plot points and draw regression line
  plot(x, y, xlab = "x",ylab="y", pch = 16, col=4)
  abline(lmcoef <- lm(y~x)$coef, col=1)
}
redraw(x,y)
update1$AddCallback("clicked", function(...){
  xy <- getNewVals()
  redraw(xy$x,xy$y)})

```

## 3 Extensions

### 3.1 More widgets

Several other types of widgets will be added to illustrate various capabilities of RGtk. First, check buttons will be added to allow the user to choose which lines will be plotted (least squares and/or a robust fit). Each check box has its own callback, which will be read by the redraw function.

```

## Button to choose lm fit
checklm <- gtkCheckButtonNewWithLabel("Show Least Squares Fit",FALSE)
checklm$SetUsiZe(20,20)
## Button to choose resistant fit
require(MASS) ## (Venables and Ripley, 2002)
checkrlm <- gtkCheckButtonNewWithLabel("Show Resistant Fit", FALSE)
checkrlm$SetUsiZe(20,20)
checklm$SetActive(TRUE) ## Set lm fit to be drawn initially
checkrlm$SetActive(FALSE) ## Set rlm fit to be undrawn initially

## Add callbacks for the toggled buttons
checklm$AddCallback("toggled", function(...){
  xy <- getNewVals()
  redraw(xy$x,xy$y)})
checkrlm$AddCallback("toggled",function(...){
  xy <- getNewVals()
  redraw(xy$x,xy$y)})
## redefine the redraw function to include robust fit
redraw <- function(x,y){
  ## function to plot points and draw regression line
  plot(y ~ x, xlab = "x",ylab="y", pch = 16, col=4)
  if( checklm$GetActive() ) abline(lm(y~x), col=1)
  if( checkrlm$GetActive() ) abline(rlm(y~x), col=2)
} ## uses R scoping to find checklm and checkrlm

```

The callbacks are activated whenever the check boxes are “toggled”, meaning whenever the user clicks them on or off. Their actions are to invoke the `redraw` function which now assesses the state of each check box in order to add – or not – the appropriate line.

Next, to illustrate the use of a sliding scale, a loess smoother will be added. (No justification will be made for pedagogical appropriateness.) Whenever the user moves the slider, the callback to the `gtkAdjustment` widget will redraw the plot. The `gtkAdjustment` function sets up the initial value of the scale, the lower and upper limits, and the increments of change per mouse-click.

```

## Define the range of values for loess.span and the increments of movement
# This allows the span to vary from 0 to 1 with increments of .1
smoothness <- gtkAdjustment(.5, 0, 1.1, .1, .1, .25) # .5 is initial value
hscale <- gtkHScale (smoothness)
hscale$SetUsiZe ( 100, 20)
# Create a callback for when the slider thumb is moved.
smoothness$AddCallback("value-changed", function(...) {
  xy <- getNewVals()
  redraw(xy$x,xy$y)})
## again, redraw must be modified, adding the line:
lines(lowess(x, y, loess.span), col=4)

```

In order to pack the new widgets and a label into the display window, I use a table rather than several boxes, as the table simplifies alignment of multiple widgets.

```
label2 <- gtkLabel( paste(
```

```

"Smoothness of the Lowess smoother goes from 0 to 1.",
"Move the slider to set the smoothness",sep="\n"), show=FALSE)
# Create table to hold Check Buttons and lowess-smoothness slider
table1 <- gtkTable(3,2, homo=TRUE, show=FALSE)
table1$Attach(checkLM, left.attach=0, right.attach=1,
top.attach=0, bottom.attach=1)          ## Top Left Cell
table1$Attach(checkRLM, left.attach=0, right.attach=1,
top.attach=1, bottom.attach=2)         ## Bottom Left Cell
table1$Attach(label2, left.attach=1, right.attach=2,
top.attach=0, bottom.attach=1)        ## Top Right Cell
table1$Attach(hscale, left.attach=1, right.attach=2,
top.attach=1, bottom.attach=2)        ## Bottom Right Cell

```

Finally, text boxes are added to output the equations of the lines. Unlike those used for  $x$  and  $y$  inputs, these are not set as editable. Because the code adds nothing new, it is not shown here. The completed demo is shown in Figure 2.

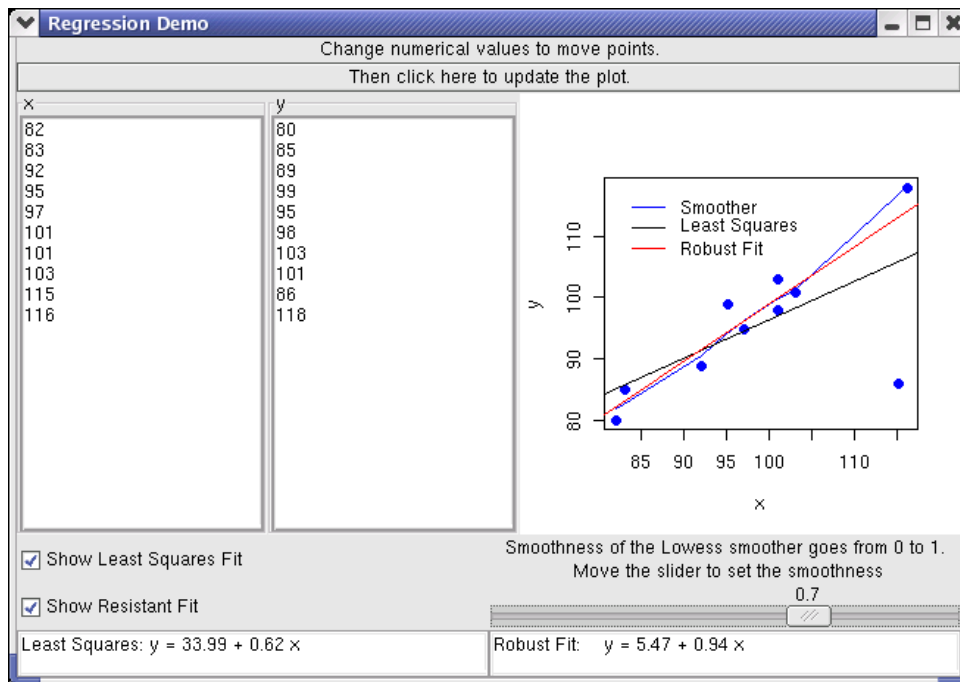


Figure 2: Screen-shot of the Completed Regression Demonstration

### 3.2 RGtkExtra

Several other R packages are available which add more features to RGtk, including RGtkPixbuf for image loading and RGtkHTML which allows addition of HTML content. See the Omegahat web pages, <http://www.omegahat.org/download/R/packages/> for a current list. The regression demo can be improved through use of the RGtkExtra package (<http://www.omegahat.org/RGtkExtra/>) which allows one to use a data sheet widget based on the gtk+extra library (

sourceforge.net/). (The package also provides icon lists and directory trees, but has the disadvantage of being unavailable for MS Windows platforms. Installing the `gtk+extra` library is non-trivial, requiring several other libraries, as documented on the `gtkextra` source pages.)

Using the data sheet, the predictor and response can be displayed as a two-column data grid. When the user changes a cell and presses enter (or a motion arrow), a callback performs the same updates as before. The code below replaces the definition of the text boxes for  $x$  and  $y$  and the `getNewVals` function above.

```

if(require(RGtkExtra)){ ## check to see if RGtkExtra is available
  ## adapted from dataViewer code in RGtkViewers package by D. Temple Lang
  sheet <- gtkSheetNew(rows=nrow(reg.frame), cols=2, show = FALSE)
  sheet$ColumnButtonAddLabel(0, "x") ## First column label
  sheet$ColumnButtonAddLabel(1, "y") ## Second column label
  for (i in 1:length(x)) { ## insert data
    sheet$SetCellText(i - 1, 0, as.character(x[i]))
    sheet$SetCellText(i - 1, 1, as.character(y[i]))
  }
  getNewVals <- function(){
    for(i in 1:length(x)){
      newVal <- as.numeric(sheet$CellGetText(i-1, 0))
      x[i] <- newVal
      newVal <- as.numeric(sheet$CellGetText(i-1, 1))
      y[i] <- newVal
    }
    return(data.frame(x = x,y = y))
  }

  sheet$AddCallback("set-cell", function(sheet, i, j) {
    ## Function to replot points and redraw the regression line
    xy <- getNewVals()
    redraw(xy$x,xy$y)
  })
  box2$PackStart(sheet)
}
else{## can't use RGtkExtra
  ## insert text box code from earlier version here instead
}

```

Notes: The cell indices returned by GTK are 0-based, so add 1 to use them as indices in R. One callback for the entire sheet is used, rather than one for  $x$ , and another for  $y$ .

The demo using a data sheet is shown in Figure 3.

## 4 Getting help

`RGtk` and the extended libraries do not have help pages for each function because the R functions are merely wrappers to C routines of GTK. The documentation for the GTK routines are the official sources. To find out what widgets are available,

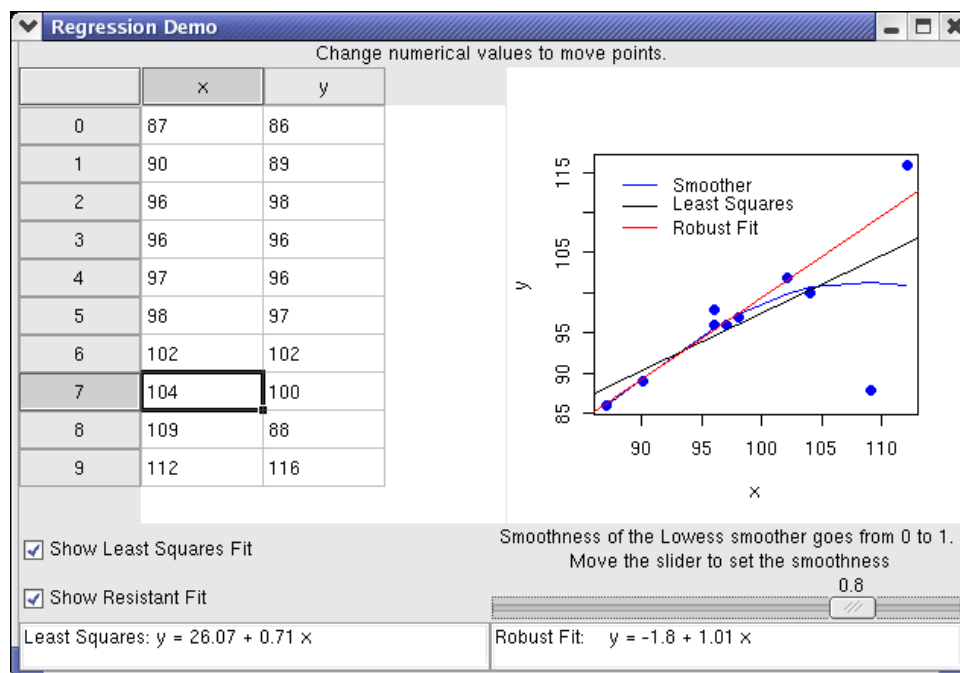


Figure 3: Screen-shot of Regression Demo Using Data Sheet

one can use one of the books about GTK [Martin (2000), Pennington (1999), Logan (2001), Wright (2000), Griffith (2000)]. Or help is available for GTK routines on the web. The GTK site offers a reference manual, <http://developer.gnome.org/doc/API/gtk/index.html>, and a tutorial, <http://www.gtk.org/tutorial/>. Some of the widgets are not currently documented. The developers' advice is to read the headers of those widgets to find out what inputs are needed and what outputs are available. That advice (and the tutorial in general) assumes one can read and understand C code, but other choices are available. Bindings have been created to interface Python with GTK. The pygtk tutorial (<http://www.moeraki.com/pygtktutorial/pygtktutorial/>) is much easier to translate into R than the C API version. For Perl users, a useful tutorial is <http://personal.riverusers.com/~swilhelm/gtkperl-tutorial/>.

Once a GTK widget has been identified, one needs to know the names of the R functions which invoke it (using `ls`) and the arguments expected (using `args`). In the following example, all functions associated with `CheckButton` are listed, and arguments for one of the functions are shown.

```
> ls("package:RGtk", patt="CheckButton")
[1] "gtkCheckButton"          "gtkCheckButtonNew"
[3] "gtkCheckButtonNewWithLabel"
> args(gtkCheckButtonNewWithLabel)
function (label, show = TRUE, .flush = TRUE)
```

## Acknowledgments

Thanks to Duncan Temple Lang for developing the RGtk bindings and answering many questions.

## References

- Peter Dalgaard. A primer on the R-Tcl/Tk package. *R News*, 1(3):27–31, 2001.
- Lyndon Drake, Martin Plummer, and Duncan Temple Lang. `gtkDevice` reference manual. <http://cran.r-project.org/doc/packages/gtkDevice.pdf>, 4 August 2003.
- Arthur Griffith. *GNOME/GTK+ Programming Bible*. Wiley, 2000.
- Ross Ihaka and Robert Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5:299–314, 1996.
- Duncan Temple Lang. The `rgtk` package. <http://www.omegahat.org/RGtk/index.html>, 4 August 2003.
- Duncan Temple Lang. The `rgtkextra` package. <http://www.omegahat.org/RGtkExtra/index.html>, 4 August, 2003.
- Syd Logan. *Gtk+ Programming in C*. Prentice Hall, 2001.
- Donna Martin. *Teach Yourself Gtk+ Programming in 21 Days*. SAMS, Inc, 2000.
- Jamie D. Mills. Using computer simulation methods to teach statistics: A review of the literature. *Journal of Statistics Education*, 10(1), 2002.
- Havoc Pennington. *Gtk+/Gnome Application Development*. Que, 1999.
- W. N. Venables and Brian D. Ripley. *Modern applied statistics with S*. Springer-Verlag Inc, 2002.
- Peter Wright. *Beginning Gtk+ and GNOME*. Wrox Press, 2000.

## Affiliation

James Robison-Cox  
Department of Mathematical Sciences  
Montana State University  
Bozeman, MT, 59717-2400 USA  
E-mail: [jimrc@math.montana.edu](mailto:jimrc@math.montana.edu)

Full R code can be found on the examples page of the RGtk site, <http://www.omegahat.org/RGtk/examples>