

# A GUI-Builder Approach to **grid** Graphics

Paul Murrell

October 20, 2008

Grid contains a lot of support for locating, sizing, and arranging graphical components on a device and with respect to each other. However, most of this support relies on *either* the parent object dictating both location and size (layouts) *or* the child dictating both location and size.

Some sorts of arrangements are more conveniently handled by having the parent dictate the location, but letting the child dictate the size. This is the situation for GUI-builders (software which forms arrangements of GUI components or widgets). The approach taken by (many ?) GUI-builders is to allow the user to create a parent *frame* and then *pack* widgets into this frame. The frame locates and arranges the children with the help of hints such as “place this widget at the bottom of the frame”, and the children dictate what size they would like to be.

This document describes a first attempt at such an interface for arranging Grid graphical objects.

## The "frame" grob

You can create a "frame" graphical object using the function `frameGrob()`. You must assign the result to a variable in order to pack grobs into it.

```
> gf <- frameGrob()
```

## The `packGrob()` function

Having created a frame, you can then pack other graphical objects into it using the `packGrob()` function. This function has a complex interface which allows for a variety of methods of packing graphical objects. The required arguments are:

`frame` is a "frame" object created by `grid.frame`.

`grob` is the grob to pack into the frame.

The remaining arguments specify where the grob is located in the frame and possibly how much space the grob should occupy. The frame is effectively just a layout; you can add grobs

to existing rows and columns or you can append the grob in a new row and/or column. If the grob is added to an existing row then the height of that row becomes the maximum of the new height and the previous height. If the row is new then it just gets the specified height. Similar rules apply for column widths.

`col` is the column to put the grob in. This can be 1 greater than the existing number of columns (in which case a new column is added).

`row` is like `col` but for rows.

`col.after` specifies that the grob should be put in a new column inserted between `col.after` and `col.after + 1`.

`col.before` specifies that the grob should be put in a new column inserted between `col.before` and `col.before + 1`.

`row.after` and `row.before` do what you would expect.

`side` specifies which side to append the new grob to. The valid values are "left", "right", "bottom", and "top".

`width` is the width of the row that the grob is being packed into. If this is not given then the grob supplies the width.

`height` is like `width` but for rows.

It is possible to modify this default behaviour. For example, it is possible to add a grob to a row and force that row to have the specified height by setting `force.height=TRUE` (and similarly for column widths). It is also possible to pack a graphical object into several rows or columns at once (although you cannot simultaneously affect the heights or widths of those rows and columns).

The result of this function is the modified frame, so you must assign the result to a variable.

```
> gf <- packGrob(gf, textGrob("Hello frame!"))
```

## "grobwidth" and "grobheight" units

A "frame" object allows a grob to specify its size by making use of "grobwidth" and "grobheight" units. These units may, of course, be used outside of frames too so their use is described here.

Consider a simple example where I want to draw a rectangle around a piece of text. I can get the size of the piece of text from the "text" grob as follows:

```
> st <- grid.text("some text")
> grid.rect(width = unit(1, "grobwidth", st), height = unit(1,
+       "grobheight", st))
```

some text

You could do the same thing with simple `"strwidth"` and `"strheight"` units, but `"grobwidth"` and `"grobheight"` give you a lot more power. The biggest gain is that you can get the size of other objects besides pieces of text (more on that soon). Another thing you can do is provide a “reference” to a grob rather than the grob itself; you do this by giving the name of a grob. What this does is make the unit “dynamic” so that changes in the grob affect the unit. The following is a dynamic version of the previous example.

```
> grid.text("some text", name = "st")
> grid.rect(width = unit(1, "grobwidth", "st"),
+           height = unit(1, "grobheight", "st"))
```

Now watch what happens if I modify the text grob named `"st"`:

```
> grid.edit("st", gp = gpar(fontsize = 20))
```

some text

Similarly, I can change the text itself:

```
> grid.edit("st", label = "some different text")
```

some different text

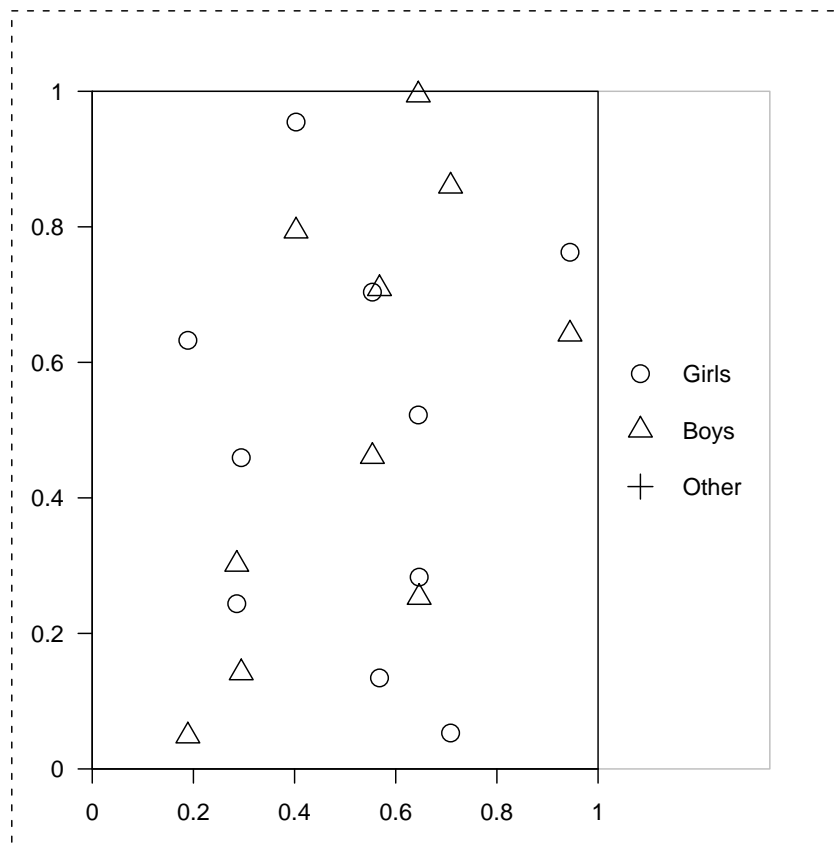
## The `widthDetails` and `heightDetails` generic functions

The calculation of `"grobwidth"` and `"grobheight"` units is a bit complicated, but fortunately most of it is automated. The simple part is that a grob provides a normal `"unit"`









## Notes

1. There are `grid.frame()` and `grid.pack()` equivalents for these functions, but these are really only useful to see the changes in the frame as each packing operation takes place.
2. This frame-and-packing stuff is easier to use, *but* (in almost all cases) it is less efficient than specifying the arrangement by hand. There is consequently a penalty to pay in terms of memory (inconsequential I think) and in terms of speed (noticeably slower).

Perhaps one sensible use of these functions is to build an image interactively using the simple arguments, which will be slow, then attempt to speed up the drawing by exploring some of the more advanced arguments.

One way to speed things up a bit is to specify the layout when the frame is initially created and then use the `placeGrob()` function to put grobs into existing rows and columns.

The speed penalty in the cases I have seen are mostly due to the time taken to generate the (sometimes very) complicated unit objects that express the heights and widths of the rows and columns of the frame layout. Future effort will be put into speeding up the creation of unit objects.