Font Families in R Graphics

Paul Murrell

July 2, 2004

The R graphics engine maintains a current font family. This font family is initialised to the value "".

All graphics devices are created with a default font. On some devices this can be controlled (e.g., via the `family` argument to `postscript()`, or the `fonts` argument to `x11()`). If the graphics engine passes a font family of "" then the device should just use its default font (this is to ensure backward compatibility with previous font handling mechanisms in R graphics).

The graphics engine font family is a simple, device-independent font name. The idea is that the user should be able to specify a simple description of a font and then individual devices must attempt to satisfy that demand. For example, users should be able to write code like the following to express the idea that text should be drawn using a monospaced font.

```
> grid.text("some text", gp = gpar(fontfamily = "mono"))
```

A graphics engine font family does not generally provide enough information on its own to accurately describe a font on a particular platform or device. The device must provide a mapping from the graphics engine font family to a full, device-specific description of a font. A suggested interface for defining and querying such font mappings has been implemented for the X11, PostScript, and PDF devices.

The X11 device provides two functions: `X11Font()` and `X11Fonts()`. The first function is used to create an X11 font description. For example, the following code creates a description of the X11 Courier font:

```
> X11Font("-*-courier-%s-%s-*-*-%d-*-*-*-*-*-*-*")

[1] "-*-courier-%s-%s-*-*-%d-*-*-*-*-*-*-*"
```

The `%s` and `%d` represent placeholders for the font weight, slant, and point size. It is not necessary to explicitly provide these because they will be inserted by the `X11Font()` function (see the example below that defines a `utopia` mapping).

The `X11Fonts()` function is used to list and (re)set the current X11 font mappings. The full list of mappings is accessed as follows:

```
> X11Fonts()
```

```
$serif
[1] "-*-times-%s-%s-*-*-%d-*-*-*-*-*-*-*"

$sans
[1] "-*-helvetica-%s-%s-*-*-%d-*-*-*-*-*-*-*"

$mono
[1] "-*-courier-%s-%s-*-*-%d-*-*-*-*-*-*-*"

$symbol
[1] "-*-symbol-%s-%s-*-*-%d-*-*-*-*-*-*-*"
```

Individual mappings can also be accessed by providing specific graphics engine font names.

```
> X11Fonts("mono", "serif")

$mono
[1] "-*-courier-%s-%s-*-*-%d-*-*-*-*-*-*-*"

$serif
[1] "-*-times-%s-%s-*-*-%d-*-*-*-*-*-*-*"
```

New mappings are defined by supplying mappings as named arguments. The name corresponds to the graphics engine font family name.

```
> X11Fonts(mono = X11Font("-*-courier-%s-%s-*-*-%d-*-*-*-*-*-*-*"),
+     utopia = X11Font("-*-utopia-*-*-*-*-*-*-*-*-*-*-*-*-*"))
```

The X11 font mappings are relatively straightforward, consisting only of a single string, but in other cases there may be more complex information stored. For example, in the PostScript case, information on font metric files and encoding files is included in the mapping.

```
> postscriptFonts("mono")

$mono
$mono$family
[1] "Courier"

$mono$metrics
[1] "com_____.afm" "cob_____.afm" "coo_____.afm"
[4] "cobo____.afm" "sy______.afm"

$mono$encoding
[1] "default"
```

For X11, PostScript, and PDF four standard mappings have been established: `"sans"` for a sans-serif font, `"serif"` for a serif font, `"mono"` for a monospaced font, and `"symbol"` for a symbol font. The PostScript device also defines some mappings for the standard Adobe fonts; the metric and encoding files for these are distributed with `R`. The PDF device makes use of the mappings set up for the PostScript device (i.e., to create new mappings or modify existing ones for the PDF device, you still use the `postscriptFont()` and `postscriptFonts()` functions).

Because of the way the PostScript device is implemented, it is necessary to declare all fonts that will be used on the device at the time the device is created. This is done via the new `fonts` argument to the `postscript()` function (see the example below). This will also be necessary when copying output from another device to a PostScript device.

For the PDF device, fonts can be declared using a `fonts` argument, just as for PostScript, but this is not necessary. Font family specifications can be given as needed (just like for an X11 device).

`R` does loads into memory all the fonts used by PostScript and PDF devices (so that calculations on font metrics can be performed quickly). This means that it is not possible to modify a font mapping during an `R` session if that font has been loaded (i.e., is being used by one or more PostScript or PDF devices).

Advantages of the new font machanism

There are three main benefits from this new mechanism:

1. It is possible to use more than one font in a plot. This is already possible on the Windows device (via `par(font=)` and the `Rdevga` file), but the new mechanism provides a consistent interface for all devices.

2. It is possible to specify different fonts within a plot in a device-neutral manner. It is not necessarily the case that all devices will produce the same output (in some cases it may be necessary to provide a new mapping for a device; in other cases, a device may just not honour a font specification), but the specification itself can be device-neutral. To see that this is actually a benefit, consider the alternative: if graphics engine font families were specified in a device-specific manner, there would be no hope at all of having a font specification work across devices.

3. The mechanism is customisable and extensible. It is easy to edit a mapping and provide a different font on a certain device for an existing graphics engine font family (e.g., change the X11 `"serif"` font to Utopia). It is also possible to specify a new mapping if a font mapping does not already exist. This may be particularly useful for device-specific cases where a special font is required to produce a plot on a particular device (e.g., use an IPA font just for a PostScript document).

Universal Font Families

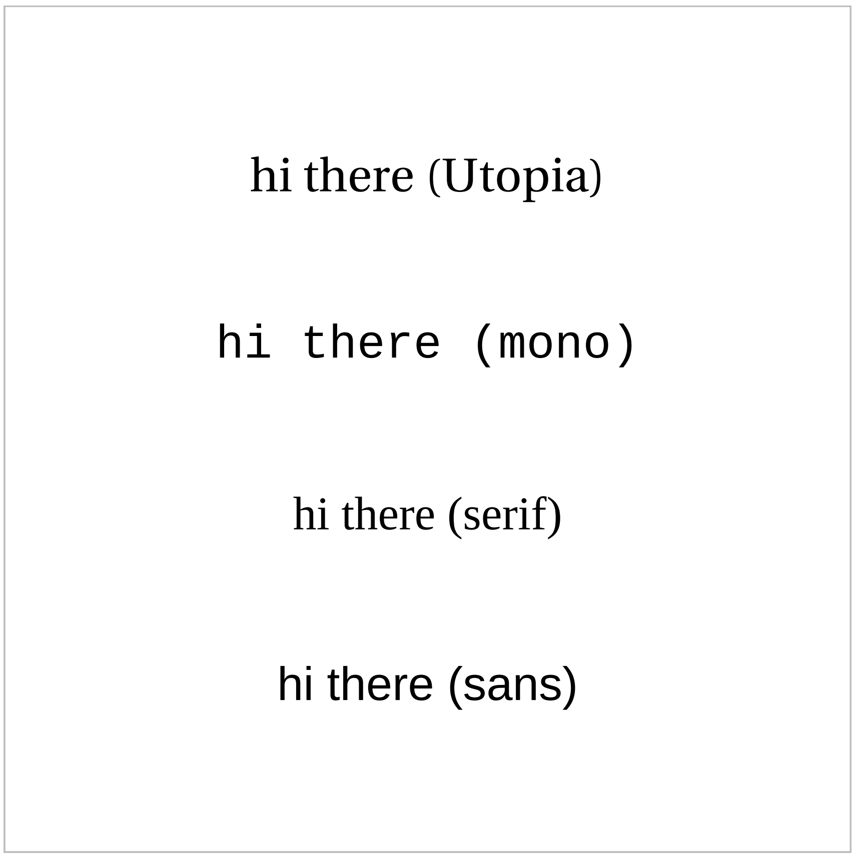
The Hershey outline fonts are available on all devices. These can be specified as the graphics engine font family, in which case the graphics engine takes care of the text drawing itself. In this case, the device doesn't draw the text, so the device does not have to worry about mappings for Hershey fonts. The following Hershey font families (and font faces) are supported (string font faces are only available via `grid`'s `gpar()` at this time):

Family	**Face (int)**	**Face (string)**
`"HersheySerif"`	1	`"plain"`
	2	`"bold"`
	3	`"italic"`
		`"oblique"`
	4	`"bold.italic"`
	5	`"symbol"`
	6	`"cyrillic"`
	7	`"cyrillic.oblique"`
	8	`"EUC"`
`"HersheySans"`	1	`"plain"`
	2	`"bold"`
	3	`"italic"`
		`"oblique"`
	4	`"bold.italic"`
`"HersheyScript"`	1	`"plain"`
	2	`"bold"`
	3	`"italic"`
		`"oblique"`
	4	`"bold.italic"`
`"HersheyGothicEnglish"`	1	`"plain"`
`"HersheyGothicGerman"`	1	`"plain"`
`"HersheyGothicItalian"`	1	`"plain"`
`"HersheySymbol"`	1	`"plain"`
	2	`"bold"`
	3	`"italic"`
		`"oblique"`
	4	`"bold.italic"`
`"HersheySansSymbol"`	1	`"plain"`
	2	`"bold"`

A PostScript example

The following example works on my Linux setup, but depends on the existence (and location) of the standard X11 Type 1 fonts.

```
> postscriptFonts(Utopia = postscriptFont("Utopia",
+     c("/usr/lib/X11/fonts/Type1/UTRG____.afm",
+         "/usr/lib/X11/fonts/Type1/UTB_____.afm",
+         "/usr/lib/X11/fonts/Type1/UTI_____.afm",
+         "/usr/lib/X11/fonts/Type1/UTBI____.afm")))
> families <- c("sans", "serif", "mono", "Utopia")
> postscript("external.ps", fonts = families)
> grid.text(paste("hi there (", families, ")", sep = ""),
+     y = 1:4/5, gp = gpar(fontfamily = families))
> dev.off()
```

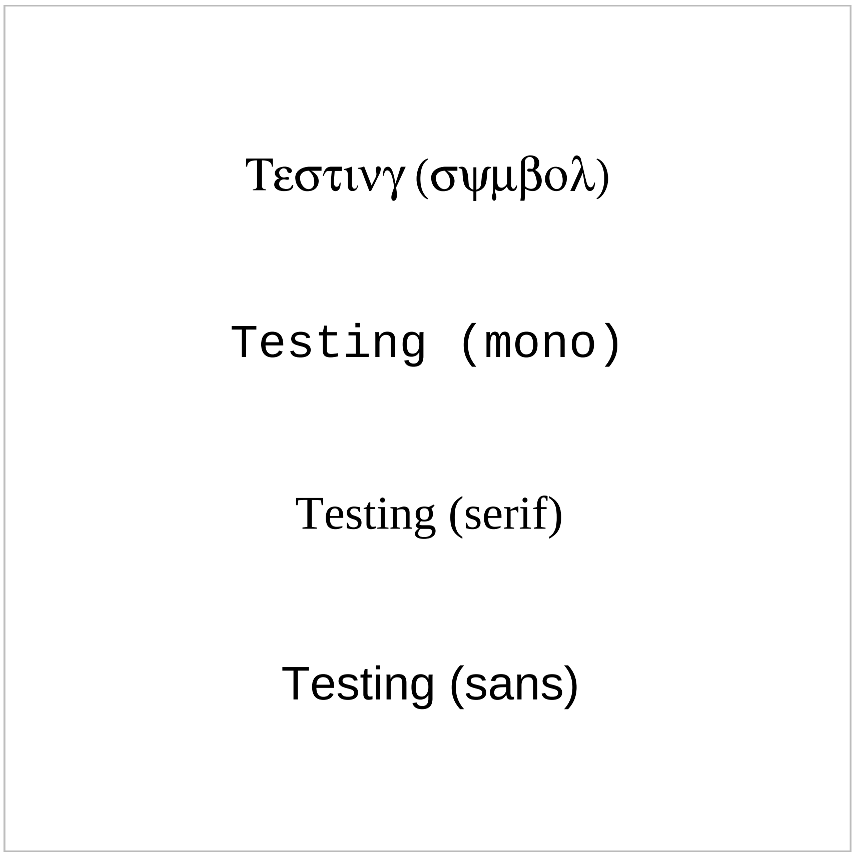


A PDF example

The following two code chunks both work for PDF.

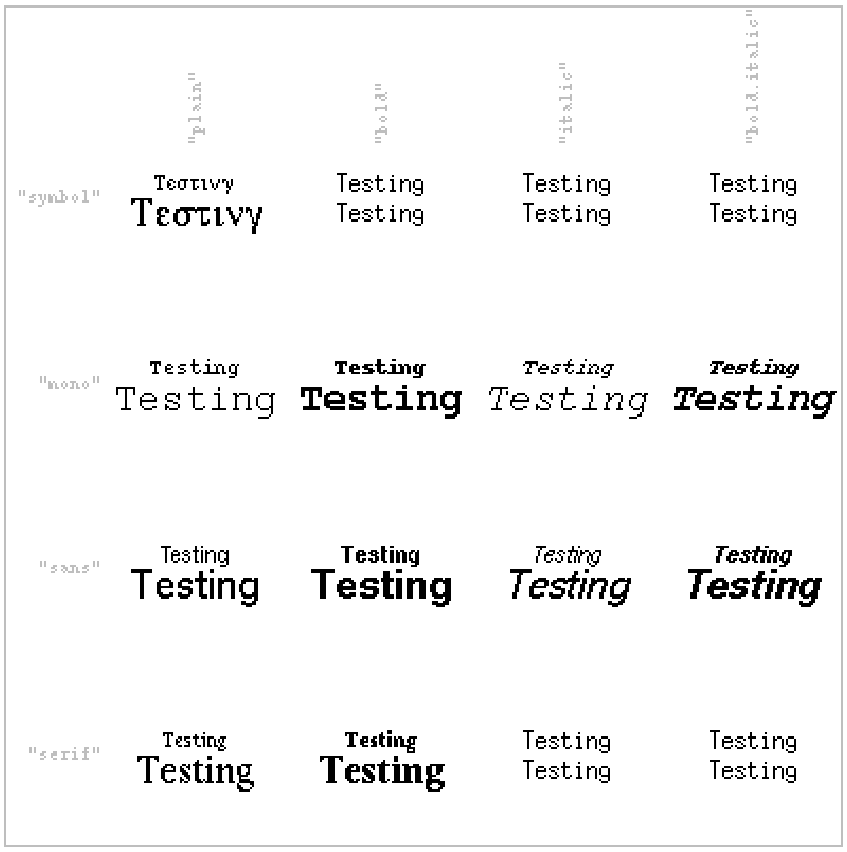
```
> families <- c("sans", "serif", "mono", "symbol")
> pdf("testing1.pdf", fonts = families)
> grid.text(paste("Testing (", families, ")", sep = ""),
+     y = 1:4/5, gp = gpar(fontfamily = families))
> dev.off()
```

```
> families <- c("sans", "serif", "mono", "symbol")
> pdf("testing2.pdf")
> grid.text(paste("Testing (", families, ")", sep = ""),
+     y = 1:4/5, gp = gpar(fontfamily = families))
> dev.off()
```



An X11 example

The X11 device doesn't do a great job of finding fonts. The following screen shot comes from testing the four different standard graphics engine font families, with the four different graphics engine font faces (`"plain"`, `"bold"`, `"italic"`, and `"bold.italic"`), at two different sizes. This works pretty poorly on my X11 server; the different fonts are found ok for the `"plain"` font face, but for the other faces the results are patchy (when the X11 device cannot find the font I ask for, it gives me the default font for the device).



Caveats

There is no requirement that graphics devices respond in any way to the graphics engine font family. Only the X11 and PostScript devices have implemented font mappings so far, but hopefully PDF, Windows and Quartz will soon follow suit. It would be nice if other devices gave a warning whenever they ignore or cannot satisfy a particular graphics engine font family specification, but it is possible that some devices will just continue to quietly do nothing.

Warning

The aim with the new mechanims is to have zero impact on existing code, but there is one place where this is definitely not true. In order to accomodate multiple fonts within a PostScript document, I had to remove the definitions of `/R`, `/B`, `/I`, `/BI`, and `/S` in the default `.ps.prolog`, so any code which relies on a custom `.ps.prolog` may break.

To Do

Apart from implementing the graphics engine font family (and font mappings) on the remaining devices, I need to implement an R-level interface for allowing the user to specify the graphics engine font family in traditional graphics (`grid` already has the `gpar(fontfamily=)` interface).

For Device Maintainers

For a start, you do not HAVE to do anything. You can just continue to ignore the graphics engine font family. As a minimum effort, you could provide a warning whenever the graphics engine font family is not `""`.

If you are maintaining a screen device, there should be not too much to do to support the graphics engine font family. You will need to provide some sort of mapping from graphics engine font family to device-specific font description, and please provide default mappings for `"serif"`, `"sans"`, `"mono"`, and `"symbol"`. Internally, you should use the mapping to set the device font within the text drawing (`dev_Text`) and font metrics (`dev_StrWidth` and `dev_MetricInfo`) device functions.

If you are maintaining a file-based device (like PostScript and PDF) there may be more work involved to keep track of which fonts are used on the device, so that these fonts can be included in the file; it will depend on how your device format allows you to specify fonts.