Font loading in LaTeX using the fontspec package: Recent updates

Will Robertson

1 Introduction

The fontspec package (`ctan.org/pkg/fontspec`) is, I am astonished to say, close to 15 years old. I honestly don't know where all the time has gone. The interface that fontspec provides was originally very simple: load a font. The font feature interface followed quickly after, and was originally based around macOS's AAT font technology, which is largely obsolete these days. Today, fontspec is targeted mainly towards the use of OpenType fonts, although the legacy AAT code is still functional. Another font technology, Graphite, can be used when running the XeTeX engine but the fontspec interface is extremely minimal (and rather undocumented).

This article will discuss some of its more recent updates, possible future interfaces, and what I now consider to be some 'best practices' for fontspec use.

2 Font loading

There are a number of ways to load or set up fonts in fontspec:

- `\fontspec`
- `\setmainfont`
- `\newfontfamily`
- `\defaultfontfeatures`

As the package has grown, the best way to combine these possibilities is probably not so clear, especially to new users.

Originally, fontspec was written for the XeTeX engine on Mac OS X, with fonts being accessed via the operating system, which provided automated interfaces for selecting, e.g., 'italic' and 'bold'. Thus, early on, users became accustomed to the idea of using the human-oriented font names known to the system, as in writing

```
\setmainfont{Hoefler Text}
```

in their document, and then everything 'just worked' without additional configuration. This feature later influenced luaotfload (for LuaTeX) to provide a similar mechanism, by scanning through known font directories and constructing its own database of font files, their 'logical' names, and the relationships between shapes within a particular family. And thus it is that, right now, you can write

```
\setmainfont{TeX Gyre Pagella}
```

in a LuaLaTeX document and the correct font will be chosen and bold and italic styles will 'just work' since the TeX Gyre fonts are included in the standard TeX distributions.

However, loading font families in these ways has confusing edge cases, and it increases document portability problems, and it can be slow. Over time I have moved away from using this feature and I think long term it would be better to de-emphasise its use. I now recommend all users to load their fonts by filename.

The most straightforward means to do this is like so:

```
\setmainfont{texgyrepagella-regular.otf}[
 ItalicFont     = texgyrepagella-italic.otf     ,
 BoldFont       = texgyrepagella-bold.otf       ,
 BoldItalicFont = texgyrepagella-bolditalic.otf,
]
```

But here there is a lot of duplication and a more streamlined way is:

```
\setmainfont{texgyrepagella}[
 Extension      = .otf         ,
 UprightFont    = *-regular    ,
 ItalicFont     = *-italic     ,
 BoldFont       = *-bold       ,
 BoldItalicFont = *-bolditalic,
]
```

Whereas in the first case the mandatory argument `{texgyrepagella-regular.otf}` was a direct reference to the 'regular upright' font file, in the second case `{texgyrepagella}` is merely a shorthand name from which the filenames are constructed.

2.1 The case against loading by font name

I claimed above that there were some problems with loading fonts by their external name instead of their filename. Let's discuss these in more detail.

Edge cases. Modern font families aren't the font families as we once knew them. It's now common to purchase families with literally tens of styles, with weights from extra light to ultra black and a multitude of styles. (I recently purchased Gill Sans Nova and it arrived as 43 separate `.otf` files.) Now that we have OpenType variable fonts, families of arbitrary complexity may start to appear. Software such as luaotfload needs to use heuristics to establish the relationships between fonts, and there can be no general solution to this task. Therefore, sometimes it fails to select the correct 'bold' font in particular circumstances; therefore, sometimes I get bug reports.

Document portability. If you are loading a font by name, it has to be installed somehow on that local computer. Well, of course any font that you load has to be installed; but if it is by name it's possible for it to be installed in a(ny) number of different locations, and the way that the font loading heuristics work, there could be a(ny) number of valid names that the font can be referred to. When failure happens, is it because you haven't installed it correctly, or have other software changes made the names no longer match, or has a database not been updated when it should have, or is it a XƎTEX vs LuaTEX difference, or ...?

Whereas if a font is loaded by filename it is generally clear, or at least discoverable, where to look and what to fix if the font cannot be found. It's also easier to set up a multiple-computer environment where all your fonts are stored, say, in Dropbox or on a secure network drive, and there is no 'font installation' process anymore. As an example, I store my 'local' `texmf` tree in Dropbox and have a shell alias set up as

```
alias texmf="sudo tlmgr conf texmf \
                TEXMFHOME ’~/Dropbox/texmf’"
```

This allows me to run `texmf` on a new (or freshly updated) computer and as long as I'm connected to Dropbox I'll be able to load all the fonts I have stored in that tree.

Slow. This is an obvious one. When luaotfload scans through your font directories, it can take quite some time. Loading fonts by filename simply doesn't have this problem. Another aspect of the 'slow' problem is that from an operating system perspective it can get slower (on older computers especially) as one installs tens or hundreds of additional fonts. (Microsoft Word's WYSIWYG font menu is a good example of this.)

3 The interface for font features

The 'plain' approach to selecting OpenType font features requires the use of raw feature tags, like `+lnum` to activate 'lining numbers' or `+dlig` to activate 'discretionary ligatures'.

```
\font\x="[EBGaramond12-Regular.otf]:+lnum;+dlig"
```

OpenType has quite a number of standard feature tags, but they are not organised into a hierarchy. In the early days of the fontspec package, its feature loading interface was modelled on Apple's AAT font technology. This competitor to OpenType, quietly fading in popularity, used a keyval approach to font features instead, which mapped more naturally to a user interface. The fontspec interface to the plain example above is:

```
\fontspec{EBGaramond12-Regular.otf}[
  Numbers   = Lining        ,
  Ligatures = Discretionary ,
]
```

UltraLight Light Book Medium Semibold Bold
Heavy ExtraBold UltraBold
CnUltraLight CnLight CnBook CnMedium CnSemibold CnBold
CnHeavy CnExtraBold CnUltraBold

Figure 1: Upright 'normal' and 'condensed' weights of Gill Sans Nova.

Deco-Regular
Shadowed-Light
Shadowed-Medium
SHADOWED-OUTLN
INLINE-COND
INLINE-EXTRALT
INLINE-LIGHT
INLINE-REGULAR
INLINE-BOLD

Figure 2: Additional 'unusual' fonts in the Gill Sans Nova family.

It is also possible to change the font features for the currently-selected font using `\addfontfeatures`. A relatively recent improvement to fontspec dramatically improved the reliability and internal logic for such commands. Now, writing

```
\addfontfeatures{Numbers = OldStyle}
```

will explicitly deactivate `Numbers = Lining` before adding the new `+onum` OpenType tag. OpenType features are now provided consistently with `Off` and `Reset` variants (e.g., `Numbers = OldStyleOff` and `Numbers = OldStyleReset`) which interact properly with the feature loading logic to either forcibly deactivate a feature or to reset that particular fontspec feature to its default state.

4 Typical example

Let's now introduce a commercial font, and use this in an example of setting up a custom font from scratch; I'll use 'Gill Sans Nova' as it comes with a range of weights and styles. This font family has upright and italic fonts in both 'normal' and 'condensed' widths with a large range of weights (see Figures 1 and 2).

When using Gill Sans Nova myself, I usually want the Medium and Bold weights paired, so I created a file in my local `texmf` directory named `gill-sans-nova.fontspec`, which looks like this:

```
\defaultfontfeatures[gill-sans-nova]{
 UprightFont    = GillSansNova-Medium.otf       ,
 ItalicFont     = GillSansNova-MediumItalic.otf,
 BoldFont       = GillSansNova-Bold.otf         ,
 BoldItalicFont = GillSansNova-BoldItalic.otf   ,
}
```

This file allows me to write, without any additional options, `\setmainfont{gill-sans-nova}` or other fontspec font selection command. Additionally, I could write:

```
\newfontfamily\bookfont{gill-sans-nova}
```

and then use `\bookfont` in the setup of the typography of the document.

To use a different selection of fonts from the family, I can create another `.fontspec` file and load that in the same way.

The `\defaultfontfeatures` command isn't *required* to be in a `.fontspec` file; it could be in a class or package file, or even just pasted into a document preamble. But this approach keeps your preamble as tidy as possible if you have a series of font family definitions you will (probably) reuse between documents.

4.1 Feature possibility: font collections?

In the example above, `\newfontfamily` was briefly mentioned for creating a macro that switches the font to a pre-defined family. This is akin to extending the selections provided in LaTeX with its `\rmfamily`, `\sffamily`, and `\ttfamily` definitions.

In other contexts it may be desirable to define a single command to replace all of the standard `rm`/`sf`/`tt` fonts. One interface we could imagine here might look like

```
\setmainfont{pagella}% default ‘collection’
\setmainfont{aldus}[Collection={examples}]
```

to support a book with a series of 'examples', which are typeset in a different style than the main text. The document might look like:

```
  this is the \textsf{main} text
  % uses the default fonts

  \selectfontcollection{examples}
  this is some \textsf{example} text
  % uses fonts from the "examples" collection
```

Over the next few months (or years, if my plans go awry), I'll experiment with implementing some interfaces along these lines for user testing. Please keep an eye out; I would certainly welcome additional feedback.

5 Additional NFSS declarations

The standard '`...Font`' variants that could be loaded within fontspec have thus far been restricted to the relatively standard set of:

`UprightFont` / `ItalicFont` / `BoldFont` / `BoldItalicFont` / `SlantedFont` / `BoldSlantedFont`

(And for each of these variants both normal and small caps shapes are allowed.)

However, LaTeX's font selection scheme allows complex mappings along what it terms the 'shape' and the 'series' axes. These can be assigned using fontspec's `FontFace` option:

```
\defaultfontfeature+[gill-sans-nova]{
 FontFace = {uu}{n}{GillSansNova-UltraLight.otf},
 FontFace = {ll}{n}{GillSansNova-Light.otf      },
 FontFace = {hh}{n}{GillSansNova-Heavy.otf      },
 FontFace = {xx}{n}{GillSansNova-ExtraBold.otf },
}
```

Now, for fonts in which only a range of weights need to be defined, as here, the fontspec interface to this is a little awkward. I have been discussing the idea of a more friendly syntax with a number of users recently. On the other hand, for truly arbitrary font shapes that cannot be categorised in a standard way (e.g., the 'Deco' Gill Sans font shown in Figure 2), the more flexible `FontFace` option provides the most generic interface.

6 'Inner' emphasis and 'Strong' emphasis

By default, writing `\emph{\emph{abc}}` produces an upright 'abc' as LaTeX knows that italic text needs to switch (back) to upright text for emphasis. Some typesetters prefer instead to emphasise within italic text with small caps, and LaTeX 2_ε provides the `\eminnershape` interface for defining the behaviour of nested emphasis beyond a single level.

In fontspec, I became interested in the idea of supporting arbitrary levels of nesting. For emphasis this is presumably of limited utility, but later in this section it will become more clear why this is useful. Therefore, fontspec now supports arbitrary nesting using (say)

```
\emfontdeclare{\itshape        ,
               \upshape\scshape,
               \itshape        ,
}
```

which will ask emphasised text to switch to italic, then upright small caps, then italic small caps, as shown in Figure 3. Each command sequence separated by commas is applied successively.

Just above, we saw that fontspec provides the `FontFace` option to load a range of font weights in

Rm \emph{*Aaa* *\emph*{EEE \EMPH{*III*}}}

Figure 3: An example of nested emphasis.

Abc \strong{**Abc** \strong{**Abc** \strong{**Abc**}}}

Figure 4: An example of nested `\strong` emphases.

a single family. Inspired by HTML's `<strong>` tag, I have added `\strong` to fontspec as a sibling to `\emph`. This is the command which makes it sensible to support arbitrary nesting, given the large range of weights seen in modern font families.

To follow from the previous Gill Sans Nova example, to setup `\strong` with nesting I can write:

```
\strongfontdeclare{
  \bfseries                  ,
  \fontseries{hh}\selectfont ,
  \fontseries{xx}\selectfont ,
}
```

This will produce the results shown in Figure 4.

Note there is no `\weaken` command to change weights in the opposite direction! More seriously, I can imagine generalising the interfaces here to a pair of code-level interfaces such as `\fontseriesup` and `\fontseriesdown` to change weights along a scale.

These interfaces will be subject to change as improved methods for loading additional bold weights are developed. At the moment the `\emfontdeclare` and `\strongfontdeclare` commands are global for all fonts; there is no current way to have font-specific declarations there.

7 Custom encodings

We all know that fonts aren't perfect, and while many problems can be solved by choosing a better font, this is not always possible. The fontspec package, in concert with recent LaTeX 2$_\varepsilon$ changes around the `TU` font encoding, now allows per-font customisation for symbols and accent support. As an example, say I'm creating a poster with some Sanskrit text but I'm using a Western font; I can choose to redefine the underdot accent `\d` by loading two fonts like so:

```
\newfontfamily\sanskritfont{CharisSIL}
\newfontfamily\titlefont{Posterama}[
  NFSSEncoding=fakedotaccent
]
```

Of course, this needs some extra setup beforehand:

```
\DeclareUnicodeEncoding{fakedotaccent}{
  \input{tuenc.def}
  \EncodingCommand{\d}[1]{%
    \hmode@bgroup
      \o@lign{\relax#1\crcr\hidewidth
      \ltx@sh@ft{-1ex}.\hidewidth}%
    \egroup
  }
}
```

Then later in the document, no additional work is needed:

```
...{\titlefont    KALITA\d M}...
...{\sanskritfont KALITA\d M}...
```

The first uses the 'fake' accent; the second the real Unicode glyph.

There is a huge caveat to this, which is that LaTeX can't intercept Unicode accents. So if you have text that is written literally as '`KALITAṂ`' in the source, LaTeX can't correct this for you, and you'll end up with whatever the font gives you. This limitation could be addressed by adding a pre-processing stage to the LaTeX typesetting, but there is no built-in mechanism to do this.

8 Conclusion

While the fontspec package has undergone extensive development over the years, the core concepts are the same: load fonts in a flexible way. But I think it's fair to say that as the package has grown and the interfaces have become more complex, the interface is not as clear as it could be. Will I ever re-write fontspec entirely? Probably not. But with LaTeX3 interfaces to consider, it's probable that a slimmed-down version of fontspec will need to make an appearance somewhere or another.

◇ Will Robertson
School of Mechanical Engineering
The University of Adelaide, SA
Australia
`will.robertson (at) adelaide dot edu dot au`
`wspr.io/fontspec`