Fonts

Advanced font features with XeTeX — the fontspec package*

Will Robertson

Abstract

This paper describes the fontspec package for the XeTeX–LaTeX format. This package provides a high level interface for font selection and configuration of OpenType and other fonts.

1 Introduction

XeTeX is an extended TeX program written by Jonathan Kew, and has been introduced recently in this journal [4]. It is currently available only on Apple's Mac OS X, but there is considerable interest (and much work to be done!) in making it a cross-platform application. The main advantages it holds over its contemporaries are support for Unicode input and direct access to fonts installed in the operating system. No additional support files are necessary in order to install such fonts, which are accessed via an extended `\font` primitive. This primitive also provides access to rich font features available in either the OpenType format, or the 'Apple Advanced Typography' (AAT) format.[1] A typical example, in plain XeTeX, of choosing a font with old-style figures in these two formats is shown in example 1.

The fontspec package is an implementation of a high level interface for LaTeX users of XeTeX to access feature-rich fonts in the framework of the familiar NFSS. Furthermore, it obviates the need for custom-written font definition files required for font installation.

This paper will introduce the fontspec package with some side commentary on the advanced font formats it supports. The first half of the paper will cover the user interface, covering font installation, font selection, and font *feature* selection. The second half discusses the implementation details, shortcomings, and future of the package.

* Version 1.9 of the package was under construction while this paper was finalised; some additions may yet make it into the final version, in which case I apologise in advance for the incomplete information here. The package documentation will always be up to date, of course.

[1] The AAT format may be considered to be approximately equivalent to a combination of the Multiple Master and OpenType formats.

Example 1: Plain XeTeX OpenType and AAT font selection with old-style figures.

```
\font\fonta="Adobe Garamond Pro:+onum" at 12pt
\fonta OpenType old-style figures: 0123456789

\font\fontb="Apple Chancery:
             Number Style=Old Styles" at 12pt
\fontb AAT old-style figures: 0123456789
```

OpenType old-style figures: 0123456789

AAT old-style figures: 0123456789

2 Motivation

With XeTeX, users have easy access to a multitude of typefaces in Plain TeX. But writing the font definition files for LaTeX was cumbersome and proved a fair obstacle for day-to-day use. I started working on a solution, which was originally simply to create the `.fd` files necessary to access every Mac OS X font in LaTeX, but I quickly tired of the tedium, disenchanted with this non-general solution to the problem. Furthermore, the scope of font features provided by feature-rich fonts quickly demonstrated the NFSS[2] insufficient for the task of incorporating every permutation of font features a user might desire.

Secondly, AAT font features are accessed by referring to a specific string defined on a per-font basis, with consistency between fonts kept between 'feature codes' rather than the value of the string. For example, example 1 uses the strings 'Number Style' and 'Old Styles' to select lowercase numbers in Apple Chancery, but for Hoefler Text one would write '`Number Case=Lowercase numbers`'. Note that such a system works well in a graphical program in which font features are selected interactively from a list; each feature may be described exactly as the font designer would wish (and this is indeed an advantage for more esoteric features). Unfortunately, for a batch program like XeTeX, this flexibility is a burden.

Finally, after version 0.8, XeTeX began using the ICU renderer[3] to support OpenType font features in addition to the Mac OS X-native features it supported from the beginning. Now the poor users had to cope with not only a different system for applying font features, but also OpenType's cryptic abbreviations for them. At this time, fontspec was able to provide

[2] NFSS, the 'new font selection scheme' for LaTeX, isn't actually that new. For readers unfamiliar with its concepts, the documentation file `fntguide` reveals all [12].

[3] International Components for Unicode, `http://icu.sourceforge.net`

a unified and consistent interface to fonts and their features in both formats, and offer a few other niceties along the way.

3 Font installation

To install a font in Mac OS X, the font file must be placed in one of the computer's `Fonts` folders (user, system, or network). Once a font is installed in Mac OS X it is immediately available to XƎTEX. No additional font support files of any kind are required for it to be loaded by a `\font` command. (A LaTeX user would still need `.fd` files for loading it with classical NFSS techniques.) Ease of font access is one of XƎTEX's attractions, but no manipulation of the font properties can occur between the stages of obtaining the font and actually using it.

Compare this to the method TeX and its siblings uses, in which sub-optimal glyphs or kerning in the font may be rectified with customised font metrics and virtual fonts. In XƎTEX, it is much simpler for a *user* to install a new font, but it is less flexible if greater output quality is desired than the font alone gives and the user cannot, or may not (due to license restrictions), edit the font file itself.

It has always been a good idea, however, to use a suitably high-quality font from the beginning in order to avoid such hassles.

4 Font selection

One of the trickier topics a new user to LaTeX faces is font selection, although progress has been made with the PostScript NFSS bundle [11], and other freely available fonts, that provide simple[4] packages to select them. XƎTEX and the fontspec package make things similarly easy by referring to a font with its display name, rather than the cryptic 'Karl Berry' abbreviation, unnecessary in many cases these days.

On an individual basis, `\fontspec` selects font families:

`\fontspec[`*font features*`]{`*font name*`}`

This command loads the specified typeface and defines an NFSS family as appropriate with bold, italic, and small caps shapes (if available) for access with the familiar font-shape changing commands such as `\itshape`, `\textsc`, *etc.*[5] An example of selecting the 'Hoefler Text' family with these methods is shown in example 2.

Example 2: NFSS family selection

```
\fontspec{Hoefler Text}
This is an example typeset in Hoefler Text.
\textit{Here is italic.} \textbf{And now bold.}
{\bfseries\itshape Bold italic, of course.}
\scshape  Small caps, if available.
\itshape  And italic small caps.
\bfseries Even bold italic small caps!
```

This is an example typeset in Hoefler Text. *Here is italic.* **And now bold.** ***Bold italic, of course.*** SMALL CAPS, IF AVAILABLE. *AND ITALIC SMALL CAPS.* ***EVEN BOLD ITALIC SMALL CAPS!***

Example 3: Choosing the default font families.

```
\setromanfont{Baskerville}
\setsansfont[Scale=MatchLowercase]{Skia}
\setmonofont[Scale=MatchLowercase]{Monaco}
The \textsf{fontspec} package defines
the \verb|\fontspec| command.
```

The fontspec package defines the `\fontspec` command.

More usefully, the default document fonts (roman, sans serif, and typewriter), are chosen with the following commands, which have the same interface as `\fontspec` itself. They are `\setromanfont`, `\setsansfont`, and `\setmonofont`, and provide a more intuitive interface than such methods as

`\renewcommand\rmdefault{`*family*`}` .

Related commands are also available for specifying the text fonts for use in maths environments (*i.e.*, `\mathrm` and others).

The use of the default-font commands in example 3 also demonstrates the feature for automatic font scaling, which in this case keeps the lowercase letter heights consistent. Further explanation of the `Scale` feature occurs in section 5.1.

The fontspec package attempts to identify the accompanying small caps, bold, and italic faces for a selected font, but in the case that it fails or that more than one is available for use, they may be selected explicitly, with individual font features if desired. 'Old-fashioned' 8-bit fonts with separate small caps may be defined as a complete family in this way, as shown in example 4, which also demonstrates how a Multiple Master font instance can be conveniently defined for the bold series.

Finally, commands may be defined for efficiently switching between fonts:

[4] Indeed, too simple in some cases. The helvet package allows a scaling factor to load the font at any relative size, but almost all other font packages skip this sometimes-vital ingredient.

[5] Code inspired by Philipp Lehman's *The Font Installation Guide* [6] allows the combination of both italic and small caps shapes.

Example 4: Choosing accompanying fonts.

```
\fontspec[SmallCapsFont = {Minion MM Small Caps
                           & Oldstyle Figures},
               BoldFont = {Minion MM Roman},
           BoldFeatures = {Weight = 1.4},
         ]{Minion MM Roman}
Minion Roman 123 \\ \textsc{Minion SC 456} \\
\bfseries
Minion Bold 123 \\ \textsc{Minion Bold SC 456}
```

Minion Roman 123
Minion SC 456
Minion Bold 123
Minion Bold SC 456

`\newfontinstance\fontcs[`*features*`]{`*font name*`}`

This defines the `\fontcs` control sequence (say) for selecting the particular font instance defined. It is more efficient than writing

`\def\fontcs{\fontspec[`*features*`]{`*font name*`}}`

because the feature processing only needs to be performed in the original definition, as opposed to every time the macro is expanded in the latter case.

5 Font feature selection

The package documentation covers all of the built-in features fontspec supports, with many examples; an interesting subset of these is presented here.

As previously mentioned, font features are defined in the optional argument to the various font commands. Inspired by the organisation of the AAT font features, these are separated into groups and use keyval comma-separated values to impose some sort of structure onto the large number of possible feature choices.

Two very important commands are associated with choosing font features:

`\defaultfontfeatures{`*font features*`}`
`\addfontfeatures{`*font features*`}`

The first, `\defaultfontfeatures`, is used to define features that will be applied implicitly to *all* subsequent font choices; for example, to request that all fonts use lowercase numbers.

Secondly, the command `\addfontfeatures` (the `s` is optional) selects features to use *in addition* to those already specified for the current font. The scope of this command is local to the current group; the default fonts are not redefined. This could be used, *e.g.*, in a hook to all tabular material that selects uppercase and fixed-width numbers regardless of the font in use; refer to example 5 for a proof-of-concept

Example 5: Selecting number styles by context.

```
\fontspec[Numbers={Proportional,OldStyle}]
                                   {Skia}
In 1842, 999 people sailed 97 miles
in 13 boats. \par In 1923, 111 people
sailed 54 miles in 56 boats. \vspace{10pt}

{\addfontfeatures{Numbers={Monospaced,Lining}}
\begin{tabular}{@{} cccc @{}}
  Year & People & Miles & Boats \\
  \hline
  1842 & 999 & 75 & 13 \\
  1923 & 111 & 54 & 56 \\
\end{tabular}}
```

In 1842, 999 people sailed 97 miles in 13 boats.
In 1923, 111 people sailed 54 miles in 56 boats.

Year	People	Miles	Boats
1842	999	75	13
1923	111	54	56

implementation of this idea, which was requested by Toledo in his paper on feature-rich fonts [13].

The important point is that the tabular material requires no *a priori* knowledge of the font in use to format the numbers appropriately. The implementation of a class that takes advantage of such features is left as an exercise to the reader.

Another example of the `\addfontfeature` command is shown in example 6, in which vulgar fractions may be easily created on the fly.[6] This example also uses a macro from the xunicode package, by Ross Moore, which provides macros (backwards compatible with existing LaTeX conventions) for a large number of Unicode character slots; in this case, `\textfractionsolidus` refers to U+2044 FRACTION SLASH.

5.1 Features for all fonts

The first class of features is independent of the choice of font; that is, there are no restrictions on their use and they do not require specific font support.

The first is the most interesting, and deals with a specific feature of XeTeX. The `Mapping` feature allows a last minute Unicode remapping of the input stream, providing such features as non-standard

[6] In practise, as always, things are a bit more complex; the `\vfrac` macro shown is actually AAT-font specific due to differences with how such things are handled in OpenType. A more complex macro could be written to handle both cases transparently, however.

Example 6: On-the-fly vulgar fractions.

```
\newcommand\vfrac[2]{%
  {\addfontfeature{%
     VerticalPosition = Superior}#1}%
   \textfractionsolidus
  {\addfontfeature{%
     VerticalPosition = Inferior}#2}}

\fontspec{Hoefler Text}
 \vfrac{147}{2356}      \qquad
\fontspec{Apple Chancery}
 \vfrac{963}{8754}
```

$^{147}/_{2356}$ *$^{963}/_{8754}$*

Example 7: XƎTEX's character mapping capabilities.

```
\def\text{!```Standard'' \TeX\ ligatures
                     --- for convenience!}
\fontspec{Didot} \text \\
\fontspec[Mapping=tex-text]{Didot} \text
```

!```Standard'' TEX ligatures --- for convenience!

¡“Standard” TEX ligatures — for convenience!

Example 8: fontspec's Color feature, with transparency. Looks much nicer in colour.

```
\fontspec{Hoefler Text Black}
{\addfontfeature{Color=FF000099}W}\kern-1ex
{\addfontfeature{Color=0000FF99}S}\kern-0.8ex
{\addfontfeature{Color=DDBB2299}P}\kern-0.8ex
{\addfontfeature{Color=00BB3399}R}
```

WSPR

Example 9: Too much interword-space is the devil.

```
\fontspec{Cochin}\text\par
\fontspec[WordSpace={0.7,0.8,0.9}]{Cochin}\text
```

It is well known that the space between words in a paragraph should be small to lead the eye along the line, preventing distractions from nearby text above and below.

It is well known that the space between words in a paragraph should be small to lead the eye along the line, preventing distractions from nearby text above and below.

TEX ligatures such as ``, ---, and '' for fonts without them (see example 7), and also transliteration schemes. It holds many similarities with Ω's translation processes, although a detailed comparison between the two has not yet been made.

The `Scale` feature is useful for matching the sizes of fonts with incompatible 'x'-heights (to the extent that such a thing is possible), as shown previously in example 3. In contrast to the option seen there, `MatchUppercase` matches the heights of the *upper*case letters, instead.[7] Both of these scaling features use the current default roman font for reference. Explicit scaling values can be used for fine-tuning the results (*e.g.*, `[Scale=0.84]`). More scaling options may be introduced in the future; for example, `MatchAlphabetLength` might be useful. . . .

The `Colour` feature (also called `Color`, for the Americans) is defined as a triplet of two hex numbers in RGB space, with optional transparency as another two, as shown in example 8.

XƎTEX has more advanced colour management abilities that are manipulated with `\special`s, but they are not provided for within fontspec. An interesting property of the colour options shown above is that the colour itself is a property of the glyph, so no complexity arises trying to manage a colour stack [9], or dealing with potential formatting side-effects of the `\special` commands themselves.

The remaining font-independent features relate to the low level TEX commands for specifying the space between words and choosing the hyphenation character. `WordSpace` takes either a single scaling factor or a triplet of three to change the nominal values for the respective interword space, stretch, and shrink, for the case that a suitable value isn't chosen by default. This feature is shown in example 9.

The `PunctuationSpace` feature takes a scaling factor to adjust the amount of space added after punctuation if `\frenchspacing` is not being used. This feature and the previous provide a high-level interface to the primitive `\fontdimen`s 2, 3, 4, and 7.

`HyphenChar` selects the hyphenation character to be used, of more interest when using Unicode fonts with the possibility of stylistic variations in this area. Rather than demonstrating this feature, however, in example 10 the `Uppercase` feature is shown that both adjusts the punctuation to its uppercase form and increases the tracking between letters slightly.

[7] In XƎTEX, incidentally, the cap height of a font is stored in `\fontdimen8`.

Example 10: Selecting an uppercase hyphenchar. Rather than doing it manually, an OpenType font feature performs this automatically.

```
\fontspec{Adobe Garamond Pro}      \TEXT
\addfontfeature{Letters=Uppercase} \TEXT
```

UPPERCASE TRACKING AND HYPHEN-CHAR
UPPERCASE TRACKING AND HYPHEN-CHAR

Example 11: Ligature examples

```
\fontspec[Ligatures=NoCommon]{Hoefler Text}
  strict firefly \quad
\fontspec[Ligatures=Rare]{Hoefler Text}
  strict firefly

\fontspec{Palatino}
  Apple \quad
\fontspec[Ligatures=Logos]{Palatino}
  Apple
```

strict firefly   ſtrict firefly
Apple

5.2 Features for fonts that support them

Ligatures are single glyphs that are used to represent many characters for reasons of elegance, decoration, or tradition. They are essential for typesetting many non-Western languages correctly. Some of the ligature features are shown in example 11.

Another example of shape-specific feature selection is shown in example 12, for the case of a stylistic variant in the roman form, and contextual swashes in the italic (a decidedly dubious combination). In this example, the `Contextuals` feature controls the contextual swashes on the lowercase letters, and `Alternate=1` activates swash caps for the uppercase.

Glyph variations can go on almost indefinitely for decorative typefaces; they are enumerated within the `Variant` feature. Apple Chancery is a font distributed with Mac OS X, a very small selection of whose variant glyphs are shown in example 13. For a more extreme example, see 'Zapfino', example 15.

Features for which I haven't shown examples include: optical font sizes, fractions, various ideographic and alphabetic CJK features, including support for vertical typesetting, and OpenType script and language selection.

Example 12: Different forms for different shapes.

```
\fontspec[
  UprightFeatures = {Style = Engraved} ,
   ItalicFeatures = {
    Contextuals = {WordInitial, WordFinal} ,
      Alternate = 1 }
         ]{Hoefler Text}
[ABCD\dots WXYZ] \\
\textit{Australian vegemite}
```

[ABCD...WXYZ]
Australian vegemite

Example 13: Apple Chancery's design complexities.

```
\fontspec[Variant=1]{Apple Chancery}
ventriloquizes \\
\fontspec[Variant=2]{Apple Chancery}
ventriloquizes \\
\fontspec[Variant=3]{Apple Chancery}
ventriloquizes
```

ventriloquizes
ventriloquizes
ventriloquizes

5.3 Font feature meta-details

User commands are available to specify font features that fontspec doesn't cater for. These use the plain TeX syntax to define new keys for `\fontspec`. Coincidentally, more than one feature may be activated by a single feature key in this way.

More interestingly, font features and their options may be aliased to alternate name for abbreviation or translation purposes. For example, typing `[VerticalPosition=ScientificInferior]` would be tiresome more than once or twice a document.[8] (Of course, multiple uses of a font instance should be effected with a `\newfontinstance` as previously discussed...). In any case, it may be desirable for whichever reason to rename some features and their options, as shown in example 14.

I haven't seen this key-aliasing feature offered by other packages[9] although it is simple to implement.

[8] fontspec always takes the verbose option for default option names, in the interest in self-documentation.

[9] To be honest, I haven't really looked.

Example 14: An example of font feature/option name abbreviation.

```
\aliasfontfeature{VerticalPosition}{VertPos}
\aliasfontfeatureoption{VerticalPosition}
  {ScientificInferior}{SciInf}
\fontspec[VertPos=SciInf]{Warnock Pro}
a1 b2 c3 d4 e5 f6 g7 h8 i9 j0
```

a_1 b_2 c_3 d_4 e_5 f_6 g_7 h_8 i_9 j_0

Perhaps other package authors have more confidence in their ability to assign their keys good names.

5.4 Format incompatibilities

Features for AAT fonts and OpenType unfortunately do not overlap completely. This makes the documentation more convoluted than necessary; *e.g.*, there are nine AAT ligature options and five OpenType ones, and only the first three of each are equivalent. Nonetheless, when applying the package for a specific font, the features the font contains are generally known by the user and the situation isn't so bad.[10] I expect the AAT format to slowly fade away in the face of the industry standard OpenType, so hopefully this problem will resolve itself in time.

5.5 Problems with rich fonts in a text-only environment

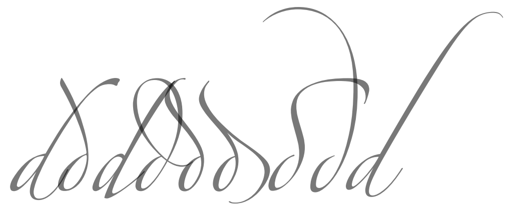
A flaw with OpenType, although it's not so much of a problem in practise, is the anæmic organisation of its features. There is just a single flat list of four letter codes, and a more structured approach like that seen in the AAT format would be easier to comprehend. AAT allows both exclusive and non-exclusive selectors: take the example of ligatures and letter shapes. More than one set of ligatures can be enabled (*e.g.*, common and rare), whereas letter shape can be only one of, say, normal or small caps or swash caps. By contrast, OpenType's features aren't partitioned into groups, and whether `clig` and `dlig` override each other or not is not obvious without trying it out or looking at the specification.

While at this stage fontspec behaves more like the OpenType model in that it is not always clear when one feature will override another, there are plans in the future to create a more robust interface that will warn or inform of such conflicts.

[10] Among the XƎTEX samples that form part of its documentation are the Plain TEX documents `AAT-info.tex` and `OpenType-info.tex`, which may be used to determine which features are supported by an arbitrary font.

Example 15: The nine variant forms of the Zapfino 'd'. How to choose between them in plain text?

```
\newcounter{var}
\whiledo{\value{var}<9}{%
  \fontspec[Variant=\thevar,
            Colour=00000088]{Zapfino}%
  d\stepcounter{var}}
```



A side-effect of having to deal with both feature formats has been to try and fit OpenType's features into the logical structure that AAT provides. Close followers of the package over time will hopefully have been not too upset when I slightly re-arranged and renamed features and options to produce a more elegant (I hope!) organisation.

For an example of the issues faced here, does a user care what the differences between 'Swashes' (OpenType feature `swsh`) and 'Contextual Swashes' (OpenType feature `cswh`) are? Is a font likely to have both of these features? In a program with a graphical user interface, a list can be presented consisting of only the features available in the current font. In a generic text editor, this is nigh on impossible. Overcoming challenges such as this could pose a significant, or at least interesting, problem.

5.6 Shorthand font feature markup

William Adams discusses the problems of markup for alternate characters in his article on using Zapfino within Ω [1]. (See example 15 for an idea of the capabilities of the same typeface in fontspec.) His proposal relies on Ω translation processes, and his scheme can be emulated with XƎTEX's font mappings.

Alan Hoenig uses more NFSS-bound techniques in his implementation of a system to use the Poetica fonts in LATEX [3]. (The fact that he was able to do this in 1995 is a testament to the underlying power of TEX.) It wouldn't be unreasonable or difficult to define commands in the spirit of Hoenig's into fontspec, such as `\+` to increment the `Variant` number of the currently selected font, but his more fanciful ideas rely on the clever structure he built into his virtual fonts.

Indeed, until now all support for feature-rich fonts have required significantly font-specific implementations, which has allowed clever shortcuts like ligatures with ^ to access alternate glyph forms (also see Toledo [13]). But since the features that fonts offer (especially 'fancy' fonts such as Apple Chancery, Poetica and Zapfino) are so varied, it is unrealistic to expect a macro package to be able to fully deal with arbitrarily complex font features in such an abbreviated manner. Provided that the font offers the support, fontspec can access any feature the designer wishes to offer without having to mess around with virtual fonts, customised ligatures, and font-specific input-to-glyph mappings, at the expense of the brevity that these schemes allow.

A solution is offered by Toledo: "A better way would be for LaTeX editors... to allow the user to select glyphs visually and to automatically produce the plain-text markup in the LaTeX input file." [13]

6 The future of fontspec

LaTeX's future involves a revamp of the font interface, in order to accommodate access to feature rich fonts through more advanced TeX-variants such as XeTeX. This package simply adds another layer on top of the NFSS, but FRANKly, I am not well-equipped enough (yet...) to implement an NFSS-replacement, either in experience or time. Hopefully, however, fontspec grows into something more suitable, or plants the seeds to get there via a different route.

On an internal level, moving the feature processing code from keyval to xkeyval is planned for the current and next release; this latter package contains many conveniences that will simplify fontspec's feature processing considerably, as well as provide some functionality that would be too convoluted to perform (for me) without it.

6.1 Moving away from the NFSS

The NFSS has served the LaTeX community extraordinarily well, especially when the fact that it is over ten years old is considered. What have been its greatest successes? The first and most obvious is its context-independent commands for shape selection. `\emph` is its crowning achievement in this area. Although many users remain blissfully unaware of this fact, writing logical markup really is next to Godliness.

A greater technical achievement (in my eyes) is the entirely transparent selection of optically-sized fonts. Granted, there haven't been that many of them. But the fact that `\fontsize{9}{11}` and `\fontsize{12}{14}` selects different shapes of Computer Modern is a testament to the cleverness of the original writers.

Example 16: Only some of the weights of Helvetica Neue; the only way to access them in the NFSS is with fixed weight codes.

```
\newcommand\showfont[1]{\fontspec{#1}#1\par}
\showfont{Helvetica Neue UltraLight}
\showfont{Helvetica Neue Light}
\showfont{Helvetica Neue}
\showfont{Helvetica Neue Condensed Bold}
\showfont{Helvetica Neue Condensed Black}
```

Helvetica Neue UltraLight
Helvetica Neue Light
Helvetica Neue
Helvetica Neue Condensed Bold
Helvetica Neue Condensed Black

Where has the NFSS fallen down? While understandable from technical aspects in the day, the small number of font axes makes it quite difficult, or even impossible, to squash every font in some of the larger font families (Lucida specifically comes to mind) into its framework. Something like the slantsc packages's support for italic small caps can only go so far (in fact, it hardly goes far enough).

Secondly, the weight/width series of the NFSS cannot be traversed in an easy manner. If a font with book, medium, and bold weights is installed, there is no convenient way to move between them, let alone condensed or expanded versions. In the same way, it is difficult for fontspec to provide access to more than the canonical bold/italic/small caps shapes defined by the `\text..` commands. Expect an improvement on this front in a future version of fontspec that uses parameterised series identifiers to deal with fonts with many weights, such as Helvetica Neue, which is shown in example 16. In the NFSS, two letter series codes would be required to move between these shapes. A relative boldness command (*cf.* the relsize package) would perform the job much more conveniently.

Finally, any *further* promulgation of font commands that look like `\textZZ`, in which `ZZ` can be any cryptic combination of letters that only made sense when it was conceived, has got to stop. There is only so much meaning that can be obtained from two-letter abbreviations, and trying to get lightface, lining figure, oldstyle figures, superior numbers, and so on into that system implies incomprehensible command names.

6.2 Limitations of fontspec

The fontspec package started out before I even knew how to program in LaTeX, but on the whole I feel that it has been quite successful. The user interface is still being refined, but it is becoming more stable. Anecdotally, people seem to have found it useful to write (often multilingual) documents in LaTeX with it. However, it is only the first step along the way to a 'future font selection scheme', and there are many improvements that could be made.

The OpenType specifications [2] define not only the corpus of standardised features, but also how interacting font features should behave. At present, features are processed by XeTeX in the order that the user specifies, but `\addfontfeatures` might override one feature with another — fontspec should be able to inform the user what is going on in this case (*i.e.*, which features are being overridden and why).

The second, larger problem is that fontspec is XeTeX-specific. A worthy and involved goal would be to unify font access based on OpenType support between the three main TeX variants: pdfTeX via the LCDF Typetools [5], which 'only' provide the means to generate readable fonts based on OpenType font features, not a higher-level interface or consistent framework for doing so; Ω and its future descendants when they emerge [7, 10]; and XeTeX as discussed herein. $\varepsilon_{\mathcal{X}}$TeX is another TeX-like program that I haven't even considered yet, but whose organisation and development seems very promising [8]. For now, fontspec's implementation is closely tied to the syntax shown in example 1; abstracting the `\fontspec` command away from XeTeX is not planned for the short term, however.

The most applicable work in this direction would be for the pdfTeX–LaTeX format. It is a great advantage to be able to refer to fonts by name, and an extra file in the TDS that contains a mapping between logical font names and 'Karl Berry'–like font filenames could be used to achieve this. Hypothetically, fonts would keep their current obscure abbreviated names for selection in plain pdfTeX; for a fontspec-like higher-level implementation, this new file (or collection of files) could be `\input` and processed during the TeX run to get from, *e.g.*, `[Numbers=Lowercase]{Aldus}` to `pasj`.

7 Implementation notes

LaTeX's NFSS defines font families with variations in series and shape only. With some extra code, it can be extended to handle small caps independently of italics. Further distinctions in more advanced fonts, such as lining and old-style figures, swash characters and inferior/superior numbers, can be organised with new sub-families by appending various characters to the original family name. These ideas have already been instantiated by Philipp Lehman, in an experimental package he calls nfssext [6].

As an example, the Aldus typeface is given the NFSS family name `pas` in its basic form, and `pasj` is Aldus with lowercase numbers. The nfssext package contains macros that switch from `pas` to `pasj`, and other forms such as `pas1` for superior numbers, that work for any collection of font families that adhere to these principles.

The fontspec package uses this idea of varying font features based on varying NFSS family names. However, it does not use the idea of manipulating the font family name directly, since combinations of variations end up creating an explosion of family names.

Rather than creating, before-the-fact, a slew of `.fd` files for each desired permutation of font features, then carefully selecting between them when it comes time to actually choose the font, fontspec instead defines new families as it goes, based on the features that are actually requested at the time.

When a font is selected, the feature list is processed to create a unique identifier for this particular font instance. This identifier is used to determine if a font has been loaded previously; at the same time, the NFSS family name is created by incrementing a counter appended to the name of the plain font.

For example, the unique identifier created when requesting 'Hoefler Text' with lining figures and coloured red could be:

```
Hoefler Text+col:CC0000+21,1
```

where $(21, 1)$ is the AAT feature code for accessing lining figures. This is used in a `\csname...\endcsname` construct to refer to the font family name; *e.g.*, `Hoefler Text (0)`, if this is the first time that Hoefler Text has been selected in any form.

Once the NFSS family name is created, it is used in *another* `\csname...\endcsname` construct to save the information used to create it; *i.e.*, the features requested and the font name, such that they can be retrieved for subsequent use with the `\addfontfeatures` command.

These steps for font selection, and later feature addition, are summarised as follows.

1. The commands

```
\defaultfontfeatures{Numbers=OldStyle}
\fontspec[Ligatures=Rare]{Warnock Pro}
```

2. are equivalent to

```
\fontspec[Numbers=OldStyle,
          Ligatures=Rare]{Warnock Pro}
```

3. which produces the font instance identifier
   ```
   Warnock Pro+onum+dlig
   ```
4. which, in `\csname...\endcsname`, gives
   ```
   Warnock Pro (0)
   ```
   (the NFSS family name)
5. which, in another `\csname...` construct, gives
   ```
   {Numbers=OldStyle,Ligatures=Rare}
   {Warnock Pro}
   ```
6. This is used by
   ```
   \addfontfeature{Color=CC0000}
   ```
7. which is in this case equivalent to
   ```
   \fontspec[Numbers=OldStyle,
             Ligatures=Rare,
             Color=CC0000]{Warnock Pro}
   ```
8. and the process starts again. So that's how fonts are selected in fontspec!

At present, the same features loaded in a different order constitutes the creation of a separate font family. Overcoming this flaw simply involves sorting the features as they are added to the font instance identifier, which shouldn't be too hard to organise, in theory.

A trickier problem is informing the user which features are available for a newly-selected font. To accomplish this, when a new font is selected, a loop would be used to build up a list of the features and options within the font. This list of (low-level) feature strings must then be converted to fontspec-level commands for display in the transcript file on request, which would mean that font feature definition in the source would have to be a two-way mapping from both AAT and OpenType features to fontspec features.

8 Conclusion

Well, I hope you've enjoyed this brief look into this XƎTEX–LATEX package. I've learned an incredible amount since I started, I've got a lot more to learn, and I'm looking forward to improving the package in the future... but not too soon. Comments, suggestions, and criticisms are all especially welcomed.

9 Acknowledgements

The fontspec package could not have been possible without the efforts of Jonathan Kew, who is developing XƎTEX at SIL and whom I thank greatly for his hard work and good advice. My gratitude goes also to those who have kindly answered my (often trivial) questions on various mailing lists and `comp.text.tex`.

References

[1] William F. Adams. There is no end: Omega and Zapfino. *TUGboat*, 24(2):183–199, November 2003.

[2] Microsoft Corporation. *OpenType Layout tag registry*. `http://www.microsoft.com/typography/otspec/featuretags.htm`.

[3] Alan Hoenig. The Poetica family: fancy fonts with TEX and LATEX. *TUGboat*, 16(3):244–252, September 1995.

[4] Jonathan Kew. XƎTEX, the Multilingual Lion: TEX meets Unicode and smart font technologies. *TUGboat*, 26(2):115–124, 2005.

[5] Eddie Kohler. *LCDF Typetools*. `http://www.lcdf.org/type/index.html#typetools`.

[6] Philipp Lehman. *The Font Installation Guide*, 2.0 edition, April 2004. (Documentation file `fontinstallationguide.pdf`.).

[7] Anish Mehta, Gábor Bella, and Yannis Haralambous. Adapting Ω to OpenType fonts. *TUGboat*, 24(3):550–556, 2003.

[8] Gerd Neugebauer et al. *The ε_χTEX project*. `http://www.extex.org/`.

[9] Heiko Oberdiek. *The pdfcolmk package*. `http://www.ctan.org/tex-archive/macros/latex/contrib/oberdiek/pdfcolmk.sty` (Documentation within `.sty` file.).

[10] John Plaice, Yannis Haralambous, Paul Swoboda, and Gábor Bella. Moving Ω to an object-oriented platform. In *Lecture Notes in Computer Science*, volume 3130, pages 17–26, 2004.

[11] Walter Schmidt. *Using common PostScript fonts with LATEX*. `http://www.ctan.org/tex-archive/macros/latex/required/psnfss/psnfss2e.pdf`.

[12] LATEX3 Project Team. *LATEX 2ε font selection*. `http://www.latex-project.org/guides/fntguide.pdf`.

[13] Sivan Toledo. Exploiting rich fonts. *TUGboat*, 21(2):121–129, June 2000.

◇ Will Robertson
School of Mechanical Engineering
University of Adelaide, SA
Australia 5005
`wspr81 (at) gmail.com`