The fontspec package

Will Robertson

2005/07/15 v1.8a

Abstract

This package provides functionality for selecting families of fonts with extended feature sets in XƎTEX-based LATEX. `\fontspec[`⟨*opts*⟩`]{`⟨*font*⟩`}` selects arbitrary fonts; `\setromanfont` sets the default for the whole document, with analogous commands for `sans` and `mono`. Other commands are provided for the advanced selection of a large (and extensible) number of rich font features.

Contents

1 Introduction

With the introduction of Jonathan Kew's XeTeX,[1] Mac OS X users can now easily access system-wide fonts directly in a TeX variant, providing a best of both worlds environment. XeTeX eliminates the need for all those files required for installing fonts (`.tfm`, `.vf`, `.map`,...) and provides an easy way to select fonts in Plain TeX: `\font\tenrm="Times New Roman" at 10pt`.

However, it was still necessary to write cumbersome font definition files for LaTeX, since the NFSS had a lot more going on behind the scenes to allow easy commands like `\bfseries` and `\emph`.

This package almost entirely eliminates this need by providing a completely automatic way to select font families in LaTeX for arbitrary fonts. Furthermore, it allows (again, almost) total control over the selection of rich font features such as number case and fancy ligatures present in many modern fonts provided with Mac OS X.

1.1 Usage

There are no options for this package: simply call it in your preamble somewhere. As of v1.6, it is necessary to explicitly request Ross Moore's xunicode package if you want access to use LaTeX's various methods for accessing extra characters and accents (for example, `\textbullet`, `\"u`, `\O`, and so on).

This is highly recommended, since macros such as `\%` will otherwise give you incorrect output. Note however that this will also break some characters in the default Computer Modern (or any other non-unicode encoded) fonts, but you're not using XeTeX to use *them*, are you!

```
\usepackage{fontspec,xunicode}
```

1.2 Warning

I still consider this package to be experimental, so I'm not ensuring backwards compatibility at all costs. I don't want to weigh the package down with old ways of doing things, so unfortunately this will mean that some old documents will need to be modified in order to compile correctly again. It'll be worth it in the long run, but you can curse at my lack of foresight as much as you wish in the meantime.

(→ v1.6: An example warning!)

Such things, and some other comments, are noted in the margin like this (←), with a red arrow if the change is relevant to the current release of the package.

1.3 About this manual

This document has been typeset with XeTeX using a variety of fonts to display various features that the package supports. You will not be able to typeset the documentation if you do not have all of these fonts, although I've used as many Mac OS X pre-installed fonts as possible. Running normal LaTeX (*i.e.*, with*out*

[1] `http://scripts.sil.org/xetex`

XƎTEX) on this file will generate the `fontspec.sty` file if this is required for some odd reason.

Many examples are shown in this manual. These are typeset side-by-side with their verbatim source code, although various size-altering commands (`\large`, `\Huge`, *etc.*) are omitted for clarity. Since the package supports font features for both AAT and OpenType fonts (whose feature sets only overlap to some extent), examples are distinguished by colour: blue and red, respectively.

2 Font selection

`\fontspec` `\fontspec[`⟨*font features*⟩`]{`⟨*font name*⟩`}` is the base command of the package, used for selecting the specified ⟨*font name*⟩ (which is the Mac OS X display name of the font) in a LATEX family. We shall look at font features later. As an example, look how easy it is to select the Hoefler Text typeface:

The five boxing wizards jump quickly.
The five boxing wizards jump quickly.
THE FIVE BOXING WIZARDS JUMP QUICKLY.
THE FIVE BOXING WIZARDS JUMP QUICKLY.
The five boxing wizards jump quickly.
The five boxing wizards jump quickly.
THE FIVE BOXING WIZARDS JUMP QUICKLY.
THE FIVE BOXING WIZARDS JUMP QUICKLY.

```
\def\pangram{The five boxing
             wizards jump quickly.}
\fontspec{Hoefler Text}
                 \pangram      \\
{\itshape        \pangram  }   \\
{\scshape        \pangram  }   \\
{\itshape\scshape\pangram  }   \\
 \bfseries       \pangram      \\
{\itshape        \pangram  }   \\
{\scshape        \pangram  }   \\
{\scshape\itshape\pangram  }
```

The fontspec package takes care of the necessary font definitions for those shapes as shown above *automatically*. Furthermore, it is not necessary to install the font for XƎTEX in any way whatsoever: every font that exists in one of Mac OS X's font folders may be accessed.

Note that this package redefines the `\itshape` and `\scshape` commands in order to allow them to select italic small caps in conjunction.

2.1 Font instances for efficiency

For cases when a specific font with a specific feature set is going to be re-used many times in a document, it is inefficient to keep calling `\fontspec` for every use. While the command does not define a new font instance after the first call, the feature options must still be parsed and processed.

`\newfontinstance` For this reason, *instances* of a font may be created with the `\newfontinstance` command, as shown in the following example:

FONT INSTANCES FOR EFFICIENCY

```
\newfontinstance\secfont[Letters=SmallCaps]{Hoefler Text}
\secfont Font instances for efficiency
```

2.2 Default font families

`\setromanfont`
`\setsansfont`
`\setmonofont`

The `\setromanfont`, `\setsansfont`, and `\setmonofont` commands are used to select the default font families for the entire document. They take the same arguments as `\fontspec`. For example:

Pack my box with five dozen liquor jugs.
Pack my box with five dozen liquor jugs.
Pack my box with five dozen liquor jugs.

```
\def\pangram{Pack my box with five
                   dozen liquor jugs.}
\setromanfont{Baskerville}
\setsansfont[Scale=0.86]{Skia Regular}
\setmonofont[Scale=0.8]{Monaco}
\rmfamily\pangram\par
\sffamily\pangram\par
\ttfamily\pangram
```

To match the families' sizes, you should use the Scale feature with trial and error until the heights of the lowercase letters are equal. (Although there are more factors involved, this can often give a reasonable result.)

2.3 Arbitrary bold/italic fonts

The automatic bold, italic, and bold italic font selections will not be adequate for the needs of every font: while some fonts mayn't even have bold or italic shapes, in which case a skilled (or lucky) designer may be able to chose well-matching accompanying shapes from a different font altogether, others can have a range of bold and italic fonts to chose between. The BoldFont and ItalicFont options (←) are provided for these situations. If only one of these is used, the bold italic font is requested as the default from the *new* font.

(→ v1.6: These options used to be called Bold and Italic, and these shorter names may still be used if you desire.)

Helvetica Neue UltraLight
Helvetica Neue UltraLight Italic
Helvetica Neue
Helvetica Neue Italic

```
\fontspec[BoldFont={Helvetica Neue}]
                        {Helvetica Neue UltraLight}
              Helvetica Neue UltraLight         \\
{\itshape     Helvetica Neue UltraLight Italic} \\
{\bfseries               Helvetica Neue       } \\
{\bfseries\itshape       Helvetica Neue Italic} \\
```

If a bold italic shape is not defined, or you want to specify *both* custom bold and italic shapes, the BoldItalicFont option is provided (←).

(→ v1.6: Yes, BoldItalic also works)

Optima
Optima Italic
Optima ExtraBlack
Optima ExtraBlack (italic?)
Optima Bold Italic

```
\fontspec[BoldFont={Optima ExtraBlack}]{Optima Regular}
                   Optima                          \\
{\itshape          Optima Italic                }  \\
{\bfseries         Optima ExtraBlack            }  \\
{\bfseries\itshape Optima ExtraBlack (italic?) }   \\
\addfontfeature{BoldItalic={Optima Bold Italic}}
{\bfseries\itshape Optima Bold Italic           }
```

(The \addfontfeature command, used here for brevity, is described in Section 3.2 on the following page.)

2.4 Math(s) fonts

When \setromanfont, \setsansfont and \setmonofont are used in the preamble, they also define the fonts to be used in maths mode inside the \mathrm-type commands. This only occurs in the preamble because LaTeX freezes the maths fonts after this stage of the processing. The fontspec package must also be loaded after any maths font packages (*e.g.*, euler) to be successful. (Actually, it is *only* euler that is the problem.)

`\setmathrm`
`\setboldmathrm`
`\setmathsf`
`\setmathtt`

However, the default text fonts may not necessarily be the ones you wish to use when typesetting maths (especially with the use of fancy ligatures and so on). For this reason, you may optionally use those commands listed in the margin (in the same way as our other `\fontspec`-like commands) to explicitly state which fonts to use inside such commands as `\mathrm`. Additionally, the `\setboldmathrm` command allows you define the font used for `\mathrm` when in bold maths mode (which is activated with, among others, `\boldmath`).

For example, if you were using Optima with the Euler maths font, you might have this in your preamble:

```
\usepackage[mathcal]{euler}
\usepackage{fontspec,xunicode}
\setromanfont{Optima Regular}
\setmathrm{Optima}
\setboldmathrm[BoldFont=Optima ExtraBlack]{Optima Bold}
```

and this would allow you to typeset something like this:

$X \rightarrow \mathrm{X} \rightarrow \mathbf{X}$
$\boldsymbol{X \rightarrow \mathrm{X} \rightarrow \mathbf{X}}$

```
$ X \rightarrow \mathrm{X} \rightarrow \mathbf{X} $
\\\boldmath
$ X \rightarrow \mathrm{X} \rightarrow \mathbf{X} $
```

3 Selecting font features

The commands discussed so far each take an optional argument for accessing the font features of the requested font. These features are generally unavailable or harder to access in regular LaTeX. The font features and their options are described in Section 4 on the next page, but there are, however, two other commands which may be used to affect font features, and they must be introduced first.

3.1 Default settings

`\defaultfontfeatures` It is desirable to define options that are applied to every subsequent font selection command: a default feature set, so to speak. This may be defined with the `\defaultfontfeatures{`⟨*font features*⟩`}` command. This command is used, for example, to automatically colour all of the examples in this document. New calls of `\defaultfontfeatures` overwrite previous ones.

Some 'default' Didot 0123456789
Now green, with old-style figures: 0123456789

```
\fontspec{Didot}
 Some `default' Didot 0123456789            \\
\defaultfontfeatures{Numbers=OldStyle, Colour=009900}
\fontspec{Didot}
 Now green, with old-style figures: 0123456789
```

3.2 Changing the currently selected features

`\addfontfeatures` The `\addfontfeatures{`⟨*font features*⟩`}` command allows font features to be changed without knowing what features are currently selected or even what font is being used. A good example of this could be to add a hook to all tabular material to use monospaced numbers, as shown in the following example:

‘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

```
\fontspec[Numbers=OldStyle]{Skia Regular}
`In 1842, 999 people sailed 97 miles in
 13 boats. In 1923, 111 people sailed 54
 miles in 56 boats.'            \bigskip

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

`\addfontfeature` This command may also be executed under the alias `\addfontfeature`.

3.3 Priority of feature selection

Features defined with `\addfontfeatures` override features specified by `\fontspec`, which in turn override features specified by `\defaultfontfeatures`. If in doubt, whenever a new font is chosen for the first time, an entry is made in the transcript (`.log`) file displaying the font name and the features requested.

4 Available font features

This section covers each and every font feature catered for by this package. Some, in fact, have already be seen in previous sections. There are too many to list in this introduction, but for a first taste of what is available, here is an example of the Apple Chancery typeface:

My 1st example of
Apple Chancery

```
\fontspec[
  Colour=CC00CC,
  Numbers=OldStyle,
  VerticalPosition=Ordinal,
  Variant=2]{Apple Chancery}
My 1st example of\\ Apple Chancery
```

You can see that font features are accessed with a ⟨*feature*⟩=⟨*option(s)*⟩ interface. Multiple options may be given to any feature that accepts non-numerical input, although doing so will not always work. Some options will override others in generally obvious ways; `Numbers={OldStyle,Lining}` doesn’t make much sense because the two options are mutually exclusive, and XƎTEX will simply use the last option that is specified (in this case using `Lining` over `OldStyle`).

4.1 Different features for different font shapes

It is entirely possible that separate fonts in a family will require separate options; *e.g.*, Hoefler Text Italic contains various swash feature options that are completely unavailable in the upright shapes.

Using the `Bold-`, `Italic-`, and `BoldItalicFeatures` options, separate font features may be *added* to those shapes. As a trivial example, let’s colour each shape differently:

0 – Normal
1 – Italic
2 – Bold
3 – Bold Italic

```
\fontspec[
   Numbers=Lining,
   BoldFeatures={Colour=AA0000},
   ItalicFeatures={Colour=008833},
   BoldItalicFeatures={Colour=AA8833},
         ]{Hoefler Text}
                   0 -- Normal   \\
{\itshape          1 -- Italic}  \\
{\bfseries         2 -- Bold}    \\
{\bfseries\itshape 3 -- Bold Italic}
```

Combined with the options for selecting arbitrary *fonts* for the different shapes, these separate feature options allow the selection of arbitrary weights in (at least) the Skia typeface:

Skia Regular
Skia 'Bold'

```
\fontspec[BoldFont={Skia Regular},
 BoldFeatures={Weight=2}]{Skia Regular}
Skia Regular \\ \bfseries Skia `Bold'
```

4.2 Font independent options

These features may be used with any font. `Scale` takes a single numeric argument for scaling the font glyphs, and was demonstrated in Section 2.2 on page 3. `Mapping` enables an XeTeX text-mapping scheme.

"¡A small amount of—text!"

```
\fontspec[Mapping=tex-text]{Cochin}
``!`A small amount of---text!''
```

`Colour` (or `Color`), also shown in Section 3.1 on page 5 and Section 4 on the preceding page, uses XeTeX font specifications to set the colour of the font. The colour is defined as a triplet of two-digit Hex RGB values, with optionally another value for the transparency (where `00` is completely transparent and `FF` is opaque.)

WSPR

```
\fontsize{48}{48}
\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}
```

XeTeX has more advanced colour management abilities, but they cannot be accessed through this package (yet, if at all).

4.3 Optical font sizes

Optically scaled fonts thicken out as the font size decreases in order to make the glyph shapes more robust (less prone to losing detail) which improves legibility. Conversely, at large optical sizes the serifs and other small details may be more delicately rendered. This is how type was set back in the day of lead and presses.

Optically sized fonts can be seen in either OpenType or Multiple Master varieties. The differences when dealing with these two are quite significant. OpenType fonts with optical scaling will exist in several discrete sizes, and these will be selected by XeTeX *automatically* determined by the current font size. The `OpticalSize` option may be used to specify a different optical size.

For the OpenType font Warnock Pro, we have three optically sized variants: caption, subhead, and display. With `OpticalSize` set to zero, these are selected as with a regular font:

Warnock Pro optical sizes
Warnock Pro optical sizes
Warnock Pro optical sizes
Warnock Pro optical sizes

```
\fontspec[OpticalSize=0]{Warnock Pro Caption}
 Warnock Pro optical sizes                 \\
\fontspec[OpticalSize=0]{Warnock Pro}
 Warnock Pro optical sizes                 \\
\fontspec[OpticalSize=0]{Warnock Pro Subhead}
 Warnock Pro optical sizes                 \\
\fontspec[OpticalSize=0]{Warnock Pro Display}
 Warnock Pro optical sizes
```

Automatic OpenType optical scaling is shown in the following example, in which we've scaled down some large text in order to be able to compare the difference for equivalent font sizes: (this gives the same output as we saw in the previous example for Warnock Pro Display)

Automatic optical size
Automatic optical size

```
\fontspec{Warnock Pro}
 Automatic optical size                  \\
\scalebox{0.4}{\Huge
 Automatic optical size}
```

Multiple Master fonts, on the other hand, are parameterised over orthogonal font axes, allowing continuous selection along such features as weight, width, and optical size (see Section 4.17 on page 15 for details). Whereas an OpenType font will have only a few separate optical sizes, a Multiple Master font's optical size can be specified over a large range. Unfortunately, this flexibility makes it harder to create an automatic interface through LaTeX, and the optical size for a Multiple Master font must always be specified explicitly.

MM optical size test
MM optical size test
MM optical size test

```
\fontspec[OpticalSize=11]{Minion MM Roman}
 MM optical size test                    \\
\fontspec[OpticalSize=47]{Minion MM Roman}
 MM optical size test                    \\
\fontspec[OpticalSize=71]{Minion MM Roman}
 MM optical size test                    \\
```

4.4 Ligatures

`Ligatures` refer to the replacement of two separate characters with a specially drawn glyph for functional or æsthetic reasons. For AAT fonts, you may choose from any combination of `Required`, `Common`, `Rare` (or `Discretionary`), `Logos`, `Rebus`, `Diphthong`, `Squared`, `AbbrevSquared`, and `Icelandic`.

The first three are also supported in OpenType fonts, which may also use `Historical` and `Contextual`. To turn a ligature option *off*, prefix its name with `No`: *e.g.*, `NoDiphthong`.

strict firefly
strict firefly

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

Rare: Ð Þ ð þ
Logos:
Rebus: ‰
Diphthong: Æ Œ æ œ

```
\fontspec
  [Ligatures={Rare,Logos,Rebus,Diphthong}]
  {Palatino}
Rare: Dh Th dh th                \\
Logos: apple                     \\
Rebus: \%0                       \\
Dipht\null hong: AE OE ae oe
```

Some other Apple AAT fonts have those 'Rare' ligatures contained in the `Icelandic` feature. Notice also that the old TEX trick of splitting up a ligature with an empty brace pair does not work in XƎTEX; you must use a 0 pt kern or `\hbox` (*e.g.*, `\null`) to split the characters up.

4.5 Letters

(→ v1.6: This feature has changed names along with its options, **breaking** backwards compatibility!)

The `Letters` feature (←) specifies how the letters in the current font will look. For AAT fonts, you may choose from `Normal`, `Uppercase`, `Lowercase`, `SmallCaps`, and `InitialCaps`.

THIS SENTENCE NO VERB
this sentence no verb
This Sentence No Verb

```
\fontspec[Letters=Uppercase]{Palatino}
 THIS Sentence no verb            \\
\fontspec[Letters=Lowercase]{Palatino}
 THIS Sentence no verb            \\
\fontspec[Letters=InitialCaps]{Palatino}
 THIS Sentence no verb
```

OpenType fonts have some different options: `SmallCaps`, `PetiteCaps`, `SMALLCAPS`, `PETITECAPS`, and `Unicase`. Petite caps are smaller than small caps. Mixed case commands turn lowercase letters into the smaller caps letters, whereas uppercase options turn the capital letters to the smaller caps (good, *e.g.*, for applying to already uppercase acronyms like 'NASA'). Finally, unicase is a weird hybrid of upper and lower case letters.

THIS SENTENCE NO VERB
THIS SENTENCE no verb

```
\fontspec[Letters=SmallCaps]{Warnock Pro}
 THIS SENTENCE no verb              \\
\fontspec[Letters=SMALLCAPS]{Warnock Pro}
 THIS SENTENCE no verb
```

4.6 Numbers

The `Numbers` feature defines how numbers will look in the selected font. For both AAT and OpenType fonts, they may be a combination of `Lining` or `OldStyle` and `Proportional` or `Monospaced` (the latter is good for tabular material). The synonyms `Uppercase` and `Lowercase` are equivalent to `Lining` and `OldStyle`, respectively. The differences have been shown previously in Section 3.2 on page 5.

For OpenType fonts, there is also the `SlashedZero` option which replaces the default zero with a slashed version to prevent confusion with an uppercase 'O'.

0123456789 Ø123456789

```
\fontspec[Numbers=Lining]{Warnock Pro}
 0123456789
\fontspec[Numbers=SlashedZero]{Warnock Pro}
 0123456789
```

4.7 Swashes

`Swashes` are fancy ornaments on letters. The options for AAT are `WordInitial`, `WordFinal`, `LineInitial`, `LineFinal`, and `Inner` (also known as ‘non-final’). As non-exclusive selectors, like the ligatures, you can turn them off by prefixing their name with `No`.

where is all the vegemite

```
\fontspec
    [Swashes={WordInitial,WordFinal}]
                         {Hoefler Text Italic}
where is all the vegemite
```

‘Inner’ ſwaſhes can *ſometimes*
contain the archaic long s.

```
\fontspec[Swashes=Inner]{Hoefler Text}
 `Inner' swashes can \emph{sometimes}    \\
 contain the archaic long~s.
```

For OpenType fonts, all but the `LineInitial` and `Inner` features are also supported, and `Contextual` turns on contextual swashes.

Without Contextual Swashes
With Contextual Swashes; cf. W C S

```
\fontspec{Warnock Pro}\itshape
 Without Contextual Swashes              \\
\fontspec[Swashes=Contextual]{Warnock Pro}
 With Contextual Swashes; cf. W C S
```

4.8 Diacritics

Diacritics refer to characters that include extra marks that usually indicate pronunciation; *e.g.*, accented letters. You may either choose to `Show`, `Hide` or `Decompose` them in AAT fonts.

Some fonts include `O/` *etc.* as diacritics for writing Ø. You'll want to turn this feature off (imagine typing `hello/goodbye` and getting ‘helløgoodbye’ instead!) by decomposing the two characters in the diacritic into the ones you actually want. I would recommend using the proper TEX input conventions for obtaining such characters instead.

Ó Ö Ø
O´ O¨ O/
Better: Ó Ö Ø

```
\fontspec[Diacritics=Show]{Palatino}
 O ´  \quad  O ¨  \quad  O/ \par
\fontspec[Diacritics=Decompose]{Palatino}
 O ´  \quad  O ¨  \quad  O/ \par
Better: \'O \"O \O % (requires xunicode)
```

The `Hide` option is for Arabic-like fonts which may be displayed either with or without vowel markings.

No options for OpenType fonts.

4.9 Vertical position

Some subscript (`Superior`) and superscript (`Inferior`) numbers and letters (and a small amount of punctuation, sometimes) are available in various fonts. The `Ordinal` feature is (supposed to be) contextually sensitive to only raise characters that appear directly after a number.

Normal $^{\text{superior}}$ $_{\text{inferior}}$
1$^{\text{st}}$ 2$^{\text{nd}}$ 3$^{\text{rd}}$ 4$^{\text{th}}$ 0$^{\text{th}}$ 8$^{\text{abcde}}$

```
\fontspec{Skia Regular}
 Normal
\fontspec[VerticalPosition=Superior]{Skia Regular}
 Superior
\fontspec[VerticalPosition=Inferior]{Skia Regular}
 Inferior                  \\
\fontspec[VerticalPosition=Ordinal]{Skia Regular}
 1st 2nd 3rd 4th 0th 8abcde
```

OpenType fonts also have the option `ScientificInferior` which extends further below the baseline than `Inferiors`, as well as `Numerator` and `Denominator` for creating arbitrary fractions (see next section). Beware, the `Ordinal` feature will not work correctly for all OpenType fonts!

Sup: $^{\text{abdehilmnorst (-\$12,345.67)}}$
Numerator: ^{12345}
Denominator: $_{12345}$
Scientific Inferior: $_{12345}$
'Ordinals': 1$^{\text{s}}$t 2$^{\text{n}}$d 3$^{\text{r}}$d 4$^{\text{t}}$h 0$^{\text{t}}$h

```
\fontspec[VerticalPosition=Superior]{Warnock Pro}
 Sup: abdehilmnorst (-\$12,345.67)                 \\
\fontspec[VerticalPosition=Numerator]{Warnock Pro}
 Numerator: 12345                                  \\
\fontspec[VerticalPosition=Denominator]{Warnock Pro}
 Denominator: 12345                                \\
\fontspec[VerticalPosition=ScientificInferior]{Warnock Pro}
 Scientific~Inferior: 12345                        \\
\fontspec[VerticalPosition=Ordinal]{Warnock Pro}
 `Ordinals': 1st 2nd 3rd 4th 0th
```

4.10 Fractions

(→ v1.7: This feature has changed: no backwards compatibility!)

Many fonts come with the capability to typeset various forms of fractional material. This is accessed in fontspec with the `Fractions` feature, which may be turned `On` or `Off` in both AAT and OpenType fonts. (←)

In AAT fonts, the 'fraction slash' or solidus character, which may be obtained by typing '⌥⇧ 1', is (supposed) to be used to create fractions. When `Fractions` are turned `On`, then (supposedly) only pre-drawn fractions will be used.

½ 5⁄6
1/2 5/6

```
\fontspec[Fractions=On]{Palatino}
 1/2 \quad 5/6 \\ % fraction slash
 1/2 \quad 5/6    % regular  slash
```

Using the `Diagonal` option (AAT only), the font will attempt to create the fraction from superscript and subscript characters. This is shown in the following example by Hoefler Text, whose fraction support may actually not be turned off.

13579⁄24680
13579/24680

```
\fontspec{Hoefler Text}
       13579/24680 \\ % fraction slash
\quad 13579/24680    % regular  slash
```

OpenType fonts simply use a regular text slash to create fractions:

1/2 1/4 5/6 13579/24680
½ ¼ ⅚ 13579/24680

```
\fontspec{Hiragino Maru Gothic Pro W4}
 1/2 \quad 1/4 \quad 5/6 \quad 13579/24680 \\
\addfontfeature{Fractions=On}
 1/2 \quad 1/4 \quad 5/6 \quad 13579/24680 \\
```

Some (Asian fonts predominantly, perhaps) also provide for the `Alternate` feature:

1/2 1/4 5/6 13579/24680
$\frac{1}{2}$ $\frac{1}{4}$ $\frac{5}{6}$ 13579/24680

```
\fontspec{Hiragino Maru Gothic Pro W4}
 1/2 \quad 1/4 \quad 5/6 \quad 13579/24680 \\
\addfontfeature{Fractions=Alternate}
 1/2 \quad 1/4 \quad 5/6 \quad 13579/24680 \\
```

You may wish to use a command like the following to build arbitrary vulgar fractions in fonts with numerator and denominator glyphs if the font doesn't support it explicitly:

There's a reason they're called 'vulgar': 13579⁄24680

```
\newcommand*\vulgarfraction[2]{%
  {\addfontfeature{VerticalPosition=Numerator}#1}%
   \textfractionsolidus
  {\addfontfeature{VerticalPosition=Denominator}#2}}
\fontspec{Warnock Pro}
There’ s a reason they’ re called ‘vulgar’ :
\vulgarfraction{13579}{24680}
```

(Requires the xunicode package for `\textfractionsolidus`.)

4.11 AAT Character Alternates

Alternates are tricky, because there's no definition for what they can be. This is where the swashes are in Hoefler Text, but you can't guarantee that this is where they'll always be or that they'll use the same number. So unfortunately selection of Character Alternates must be done numerically on a per font basis (although Section 5 on page 16 may help a little in this regard).

Sphinx Of Black Quartz, Judge My Vow
Sphinx Of Black Quartz, Judge My Vow

```
\fontspec[Alternate=0]{Hoefler Text Italic}
 Sphinx Of Black Quartz, {\scshape Judge My Vow} \\
\fontspec[Alternate=1]{Hoefler Text Italic}
 Sphinx Of Black Quartz, {\scshape Judge My Vow}
```

For an OpenType font, stylistic alternates used to be accessed through here, but now they're through the `Style` feature (←).

(→ v1.7: Yes, I don't have a high regard for backwards compatibility at the moment. Sorry.)

4.12 Variants

The `Variant` feature takes a single numerical input for choosing different alphabetic shapes. Don't mind my fancy example :) I'm just looping through the nine (!) variants of Zapfino.

```
\Huge
\newcounter{var}\newcounter{trans}
\raisebox{0pt}[\height][0pt]{%
  \whiledo{\value{var}<9}{%
    \stepcounter{trans}%
    \fontspec[Variant=\thevar,
      Colour=005599\thetrans\thetrans]{Zapfino}%
    \makebox[0.75\width]{d}%
    \stepcounter{var}}}
```

For OpenType fonts, `Variant` selects a Stylistic Set, specified numerically between one and twenty. I don't have a font to demonstrate this feature with, unfortunately.

4.13 Style

(→ v1.7: The old name, StyleOptions, still works.)

The options of the Style feature (←) are defined in AAT as one of the following: Display, Engraved, IlluminatedCaps, Italic, Ruby,[2] TitlingCaps, or TallCaps.

[ABCD...WXYZ]

```
\fontspec[Style=Engraved]{Hoefler Text}
 [ABCD\dots WXYZ]
```

(→ v1.7: Note the difference with the Alternate feature, which is for AAT fonts.)

OpenType supported options are Alternate (←), Italic, Ruby,[2] Swash, and TitlingCaps.

K Q R k v w y
K Q R k v w y

```
\fontspec{Warnock Pro}
 K Q R k v w y                    \\
\addfontfeature{Style=Alternate}
 K Q R k v w y
```

Latin ようこそ ワカヨタレソ
Latin ようこそ ワカヨタレソ

```
\fontspec{Hiragino Mincho Pro W3}
 Latin ようこそ ワカヨタレソ        \\
\addfontfeature{Style={Italic, Ruby}}
 Latin ようこそ ワカヨタレソ
```

TITLING CAPS
TITLING CAPS

```
\fontspec{Adobe Garamond Pro}
 TITLING CAPS                     \\
\addfontfeature{Style=TitlingCaps}
 TITLING CAPS
```

(Look closely.)

4.14 Character Shape

There have been many standards for how Japanese (and other?) kanji glyphs are 'supposed' to look. Some fonts will have many alternate glyphs available in order to be able to display these gylphs correctly in whichever form is appropriate. Both AAT and OpenType fonts support the following CharacterShape options: Traditional, Simplified, JIS1978, JIS1983, JIS1990, and Expert. OpenType also supports the NLC option.

唖噛躯 妍并訝
唖噛躯 妍并訝
啞嚙軀 妍并訝

```
\fontspec{Hiragino Mincho Pro W3}
{\addfontfeature{CharacterShape=Traditional}
 唖噛躯 妍并訝 }                      \\
{\addfontfeature{CharacterShape=NLC}
 唖噛躯 妍并訝 }                      \\
{\addfontfeature{CharacterShape=Expert}
 唖噛躯 妍并訝 }
```

4.15 Text Spacing

Many Asian fonts are equipped with variously spaced characters for shoe-horning into their generally monospaced text. These are accessed through the

[2]'Ruby' refers to a small optical size, historically for Japanese kana typography

`TextSpacing` feature.[3] For now, OpenType and AAT share the same six options for this feature: `Proportional`, `FullWidth`, `HalfWidth`, `ThirdWidth`, `QuarterWidth`, `AlternateProportional`, and `AlternateHalfWidth`. AAT also allows `Default` to return to whatever was originally specified.

Japanese alphabetic glyphs (hiragana or katakana) may be typeset proportionally, to better fit horizontal measures, or monospaced, to fit into the rigid grid imposed by ideographic typesetting. In this latter case, there are also half-width forms for squeezing more kana glyphs (which are less complex than the kanji they are amongst) into a given block of space. The same features are given to roman letters in Japanese fonts, for typesetting foreign words in the same style as the surrounding text.

```
ようこそ      ワカヨタレソ      abcdef
ようこそ      ワカヨタレソ      ａｂｃｄｅｆ
ﾖｳｺｿ        ﾜｶﾖﾀﾚｿ          abcdef
```

```
\def\test{\makebox[2cm][l]{ようこそ}%
          \makebox[2.5cm][l]{ワカヨタレソ}%
          \makebox[2.5cm][l]{abcdef}}
\fontspec{Hiragino Mincho Pro W3}
{\addfontfeature{TextSpacing=Proportional}\test}\\
{\addfontfeature{TextSpacing=FullWidth}\test}\\
{\addfontfeature{TextSpacing=HalfWidth}\test}
```

The same situation occurs with numbers, which are provided in increasingly illegible compressed forms:

```
—１２３２１—
-1234554321-
-123456787654321-
-1234567890098765 4321-
```

```
\fontspec{Hiragino Mincho Pro W3}
{\addfontfeature{TextSpacing=FullWidth}
 ---12321---}\\
{\addfontfeature{TextSpacing=HalfWidth}
 ---1234554321---}\\
{\addfontfeature{TextSpacing=ThirdWidth}
 ---123456787654321---}\\
{\addfontfeature{TextSpacing=QuarterWidth}
 ---12345678900987654321---}
```

4.16 Annotation

Various Asian fonts are equipped with a more extensive range of numbers and numerals in different forms. These are accessed through the `Annotation` feature with the following options: `Off`, `Box`, `RoundedBox`, `Circle`, `BlackCircle`, `Parenthesis`, `Period`, `RomanNumerals`, `Diamond`, `BlackSquare`, `BlackRoundSquare`, and `DoubleCircle`.

```
 1  2  3  4  5  6  7  8  9
①  ②  ③  ④  ⑤  ⑥  ⑦  ⑧  ⑨
(1) (2) (3) (4) (5) (6) (7) (8) (9)
1.  2.  3.  4.  5.  6.  7.  8.  9.
```

```
\fontspec{Hei Regular}
 1 2 3 4 5 6 7 8 9                   \\
\fontspec[Annotation=Circle]{Hei Regular}
 1 2 3 4 5 6 7 8 9                   \\
\fontspec[Annotation=Parenthesis]{Hei Regular}
 1 2 3 4 5 6 7 8 9                   \\
\fontspec[Annotation=Period]{Hei Regular}
 1 2 3 4 5 6 7 8 9
```

[3] Apple seems to be adapting its AAT features in this regard (at least in the fonts it distributes with Mac OS X) to have a one-to-one correspondence with the equivalent OpenType features. Previously AAT was more fine grained, but naturally they're not documenting their AAT tables any more, so if the following features don't work for a specific font let me know and I'll try and see if anything can be salvaged from the situation.

For OpenType fonts, the only option supported is On and Off:

1 2 3 4 5 6 7 8 9
(1) (2) (3) (4) (5) (6) (7) (8) (9)

```
\fontspec{Hiragino Maru Gothic Pro W4}
 1 2 3 4 5 6 7 8 9                                \\
\addfontfeature{Annotation=On}
 1 2 3 4 5 6 7 8 9
```

I'm not sure if XƎTEX can access alternate annotation forms, even if they exist (as in this case) in the font.

4.17 AAT & Multiple Master font axes

Multiple Master and AAT font specifications both provide continuous variation along font parameters. For example, they don't have just regular and bold weights, they have any bold weight you like between the two extremes.

Weight, Width, and OpticalSize are supported by this package. Luckily Skia, which is distributed with Mac OS X, has these variable parameters, allowing for a demonstration:

Really light and extended Skia
Really fat and condensed Skia

```
\fontspec[Weight=0.5,Width=3]{Skia Regular}
 Really light and extended Skia         \\
\fontspec[Weight=2,Width=0.5]{Skia Regular}
 Really fat and condensed Skia
```

Variations along a Multiple Master font's optical size axis has been shown previously in Section 4.3 on page 7.

4.18 OpenType scripts and languages

When dealing with fonts that include glyphs for various languages, they may contain different font features for the different character sets and languages it supports. These may be selected with the Script and Language features. The possible options are tabulated in Table 1 on the next page and Table 2 on page 17, respectively.

In the following examples, the same font is used to typeset the verbatim input and the XƎTEX output. Because the Script is only specified for the output, the text is rendered incorrectly in the verbatim input. Many examples of incorrect diacritic spacing as well as a lack of contextual ligatures and rearrangement can be seen. Thanks to Jonathan Kew, Yves Codet and Gildas Hamel for their contributions towards these examples.

العربي

```
\fontspec[Script=Arabic]{Code2000}
 ال ع ر ب ي
```

हिन्दी

```
\fontspec[Script=Devanagari]{Code2000}
 हिन्दी
```

লেখ

```
\fontspec[Script=Bengali]{Code2000}
 লেখ
```

મર્યાદા-સૂચક નિવેદન

```
\fontspec[Script=Gujarati]{Code2000}
 મર્યાદા-સૂચક નિવેદન
```

നമ്മുടെ പാരമ്പര്യ

```
\fontspec[Script=Malayalam]{Code2000}
 നമ്മുടെ പാരമ്പര്യ
```

ਆਦਿ ਸਚੁ ਜੁਗਾਦਿ ਸਚੁ

```
\fontspec[Script=Gurmukhi]{Code2000}
 ਆਦਿ ਸਚੁ ਜੁਗਾਦਿ ਸਚੁ
```

தமிழ் தேடி

```
\fontspec[Script=Tamil]{Code2000}
 தமிழ் தேடி
```

רְדָתָה

```
\fontspec[Script=Hebrew]{Code2000}
 רְדָתָה
```

Arabic	Ethiopic	Limbu	Sumero-Akkadian
Armenian	Georgian	Linear B	Cuneiform
Balinese	Glagolitic	Malayalam	Syloti Nagri
Bengali	Gothic	¶Math	Syriac
Bopomofo	Greek	¶Maths	Tagalog
Braille	Gujarati	Mongolian	Tagbanwa
Buginese	Gurmukhi	Musical Symbols	Tai Le
Buhid	Hangul Jamo	Myanmar	Tai Lu
Byzantine Music	Hangul	N'ko	Tamil
Canadian Syllabics	Hanunoo	Ogham	Telugu
Cherokee	Hebrew	Old Italic	Thaana
¶CJK	¶Hiragana and Katakana	Old Persian Cuneiform	Thai
¶CJK Ideographic	¶Kana	Oriya	Tibetan
Coptic	Javanese	Osmanya	Tifinagh
Cypriot Syllabary	Kannada	Phags-pa	Ugaritic Cuneiform
Cyrillic	Kharosthi	Phoenician	Yi
Default	Khmer	Runic	
Deseret	Lao	Shavian	
Devanagari	Latin	Sinhala	

Table 1: Defined `Scripts` for OpenType fonts. Aliased names are shown in adjacent positions marked with red pilcrows (¶).

5 Defining new features

This package cannot hope to contain every possible font feature. Two commands are provided for selecting font features that are not provided for out of the box. If you are using them a lot, chances are I've left something out, so please let me know.

`\newfontfeature` `\newfontfeature{`⟨*tag*⟩`}{`⟨*feature string*⟩`}` takes two arguments: ⟨*tag*⟩ is the name to reference the new feature by when selecting it in your font; and ⟨*feature string*⟩ the string that activates the particular font feature. For example, Zapfino contains a non-standard feature 'Avoid d-collisions'. To access it with this package, you could do the following:

sockdolager rubdown

sockdolager rubdown

```
\newfontfeature{AvoidD}{Special=Avoid d-collisions}
\newfontfeature{NoAvoidD}{Special=!Avoid d-collisions}
\fontspec[AvoidD,Variant=1]{Zapfino}
 sockdolager rubdown              \\
\fontspec[NoAvoidD,Variant=1]{Zapfino}
 sockdolager rubdown
```

Abaza	German	Igbo	Kuy	Norway House Cree	Saraiki
Abkhazian	Default	Ijo	Koryak	Nisi	Serer
Adyghe	Dogri	Ilokano	Ladin	Niuean	South Slavey
Afrikaans	Divehi	Indonesian	Lahuli	Nkole	Southern Sami
Afar	Djerma	Ingush	Lak	N'ko	Suri
Agaw	Dangme	Inuktitut	Lambani	Dutch	Svan
Altai	Dinka	Irish	Lao	Nogai	Swedish
Amharic	Dungan	Irish Traditional	Latin	Norwegian	Swadaya Aramaic
Arabic	Dzongkha	Icelandic	Laz	Northern Sami	Swahili
Aari	Ebira	Inari Sami	L-Cree	Northern Tai	Swazi
Arakanese	Eastern Cree	Italian	Ladakhi	Esperanto	Sutu
Assamese	Edo	Hebrew	Lezgi	Nynorsk	Syriac
Athapaskan	Efik	Javanese	Lingala	Oji-Cree	Tabasaran
Avar	Greek	Yiddish	Low Mari	Ojibway	Tajiki
Awadhi	English	Japanese	Limbu	Oriya	Tamil
Aymara	Erzya	Judezmo	Lomwe	Oromo	Tatar
Azeri	Spanish	Jula	Lower Sorbian	Ossetian	TH-Cree
Badaga	Estonian	Kabardian	Lule Sami	Palestinian Aramaic	Telugu
Baghelkhandi	Basque	Kachchi	Lithuanian	Pali	Tongan
Balkar	Evenki	Kalenjin	Luba	Punjabi	Tigre
Baule	Even	Kannada	Luganda	Palpa	Tigrinya
Berber	Ewe	Karachay	Luhya	Pashto	Thai
Bench	French Antillean	Georgian	Luo	Polytonic Greek	Tahitian
Bible Cree	Farsi	Kazakh	Latvian	Pilipino	Tibetan
Belarussian	Finnish	Kebena	Majang	Palaung	Turkmen
Bemba	Fijian	Khutsuri Georgian	Makua	Polish	Temne
Bengali	Flemish	Khakass	Malayalam Traditional	Provencal	Tswana
Bulgarian	Forest Nenets	Khanty-Kazim	Mansi	Portuguese	Tundra Nenets
Bhili	Fon	Khmer	Marathi	Chin	Tonga
Bhojpuri	Faroese	Khanty-Shurishkar	Marwari	Rajasthani	Todo
Bikol	French	Khanty-Vakhi	Mbundu	R-Cree	Turkish
Bilen	Frisian	Khowar	Manchu	Russian Buriat	Tsonga
Blackfoot	Friulian	Kikuyu	Moose Cree	Riang	Turoyo Aramaic
Balochi	Futa	Kirghiz	Mende	Rhaeto-Romanic	Tulu
Balante	Fulani	Kisii	Me'en	Romanian	Tuvin
Balti	Ga	Kokni	Mizo	Romany	Twi
Bambara	Gaelic	Kalmyk	Macedonian	Rusyn	Udmurt
Bamileke	Gagauz	Kamba	Male	Ruanda	Ukrainian
Breton	Galician	Kumaoni	Malagasy	Russian	Urdu
Brahui	Garshuni	Komo	Malinke	Sadri	Upper Sorbian
Braj Bhasha	Garhwali	Komso	Malayalam Reformed	Sanskrit	Uyghur
Burmese	Ge'ez	Kanuri	Malay	Santali	Uzbek
Bashkir	Gilyak	Kodagu	Mandinka	Sayisi	Venda
Beti	Gumuz	Korean Old Hangul	Mongolian	Sekota	Vietnamese
Catalan	Gondi	Konkani	Manipuri	Selkup	Wa
Cebuano	Greenlandic	Kikongo	Maninka	Sango	Wagdi
Chechen	Garo	Komi-Permyak	Manx Gaelic	Shan	West-Cree
Chaha Gurage	Guarani	Korean	Moksha	Sibe	Welsh
Chattisgarhi	Gujarati	Komi-Zyrian	Moldavian	Sidamo	Wolof
Chichewa	Haitian	Kpelle	Mon	Silte Gurage	Tai Lue
Chukchi	Halam	Krio	Moroccan	Skolt Sami	Xhosa
Chipewyan	Harauti	Karakalpak	Maori	Slovak	Yakut
Cherokee	Hausa	Karelian	Maithili	Slavey	Yoruba
Chuvash	Hawaiin	Karaim	Maltese	Slovenian	Y-Cree
Comorian	Hammer-Banna	Karen	Mundari	Somali	Yi Classic
Coptic	Hiligaynon	Koorete	Naga-Assamese	Samoan	Yi Modern
Cree	Hindi	Kashmiri	Nanai	Sena	Chinese Hong Kong
Carrier	High Mari	Khasi	Naskapi	Sindhi	Chinese Phonetic
Crimean Tatar	Hindko	Kildin Sami	N-Cree	Sinhalese	Chinese Simplified
Church Slavonic	Ho	Kui	Ndebele	Soninke	Chinese Traditional
Czech	Harari	Kulvi	Ndonga	Sodo Gurage	Zande
Danish	Croatian	Kumyk	Nepali	Sotho	Zulu
Dargwa	Hungarian	Kurdish	Newari	Albanian	
Woods Cree	Armenian	Kurukh	Nagari	Serbian	

Table 2: Defined Languages for OpenType fonts. Note that they are sorted alphabetically *not* by name but by OpenType tag, which is a little irritating, really.

Of course, you may perform similar operations with OpenType features, *e.g.*, `\newfontfeature{OTSwash}{+swsh}` to activate swash forms.

`\newfeaturecode`

`\newfeaturecode{`⟨*tag*⟩`}{`⟨*feature number*⟩`}{`⟨*selector number*⟩`}` performs the same function as the last macro, but instead of using a string, it takes two arguments to define the ⟨*feature number*⟩ and ⟨*selector number*⟩ of the feature. It works only for AAT fonts. There is no mechanism yet for the package to help inform you what codes are available for a specific font, sorry.

This is XeTeX by Jonathan Kew.

```
\newfeaturecode{Swash}{17}{1}
\fontspec[Swash]{Hoefler Text Italic}
 This is XeTeX by Jonathan Kew.
```

Change History

v1.0
- General: Initial version. 1

v1.1
- General: Name change to fontspec. 1
- `\setromanfont`: Implemented (with friends). 22

v1.2
- General: Initial OpenType support. 1
  - Support for `Scale`. 31

v1.3
- General: More OpenType support. 1
  - Support for `Mapping` and `Colour`. 31
- `\defaultfontfeatures`: Implemented. 23
- `\newfeaturecode`: Implemented. 24
- `\newfontfeature`: Implemented. 24

v1.3a
- General: Bug fix for OpenType small caps. 42

v1.4
- General: Support for `Weight` and `Width` AAT features. 32
- `\AtBeginDocument`: Selects the default `\mathXX` fonts. 45
- `\defaultfontfeatures`: Name changed from `\setdefaultoptions`. 23

v1.5
- General: New options for arbitrary bold/italic shapes. 31
- `\addfontfeatures`: Implemented. 24
- `\ifEqFonts`: Added check for empty font. 30
- `\zf@fontspec`: Added code for choosing arbitrary bold/italic fonts. 25
  - Checks if the font family has already been defined. 25
  - NFSS specifiers now take the default values. 25
- `\zf@make@font@shapes`: Absorbed font-checking from `\zf@fontspec`. 27

v1.5a
- `\AtBeginDocument`: Added fix for Computer Modern maths. 45

v1.6
- General: `Bold` option aliased to `BoldFont` 31
  - `LetterCase` is now `Letters` and options changed appropriately. 42
  - `Scale` feature now updates family name. 31
  - All AAT `Fractions` features offered. 43
  - New OpenType feature: `Language` 32
  - New OpenType feature: `Script` 32
  - OpenType letters features: `PetiteCaps` and `PETITECAPS`. 42
  - OpenType ligature features: `Contextual` and `Historical`. 41
  - OpenType stylistic sets supports under the `Variant` option. 43
- `\addfontfeatures`: Removed `\relax`ing of temporary macros. 24
- `\AtBeginDocument`: Removed mathtime support since XeTeX doesn't handle virtual fonts. Why did I put it in in the first place? 45
- `\fontspec`: Removed `\zf@currfont` (unnecessary) 22
- `\ifEqStr`: Removed entirely from code. keyval `\setkeys` used instead. 30
- `\newfontfeature`: newff counter now uses LaTeX methods rather than primitive TeX. I don't know if there is any advantage to this. 24
- `\newfontinstance`: Implemented. 23
- `\setromanfont`: Changed `\rmdefault`, etc., assigning to use `\let` directly. 22
- `\zf@fontspec`: Added code for choosing arbitrary bold/italic font features. 25

Writes some info to the `.log` file 25

`\zf@get@feature@requests`: Removed the space between the comma and `\zf@options` when it's concatenated with the defaults. This fixes that bug in the big Apple Chancery example. 28

v1.7

General: `Style` feature renamed from `StyleOptions`. 44

AAT `Numbers:SlashedZero`. 42

New feature: `Annotation` 44

New feature: `CharacterShape` 44

New feature: `OpticalSize`; works with both OpenType and MM fonts. 32

New feature: `TextSpacing` 44

OpenType `Alternate Fractions` feature. 43

OpenType `Alternate` now only AAT. 43

Removed AAT check for weight / width axes (could also be Multiple Master) 32

Some `\edef`s replaced with `\def` in the font feature processing 31

`\ifEqFonts`: Removed some redundant code. 30

`\zf@define@feature@option`: Implemented for the bulk of the feature processing code. 29

`\zf@fontspec`: More documentation of the code and some extra braces removed. 25

Optional argument now mandatory. 25

`\zf@make@aat@feature@string`: Changed some `\edef`s to `\let` 30

Removed third argument; always saves the feature string in `\zf@thisfontfeature` 30

`\zf@make@feature`: Accommodation of the `\zf@thisfontfeature` change. 29

`\zf@make@font@shapes`: Changed some `\edef`s to `\let`. 27

Support for the `OpticalSize` feature. 27

`\zf@set@font@type`: Added 'MM' font type; tests true, *e.g.*, with Skia & Minion MM. Used with the `OpticalSize` feature. 26

Removed exclusivity from font type (AAT, OpenType) check, since fonts can be both. 26

Removed various `\count255`s; replaced by `\expandafter`s directly in the conditionals. 26

`\zf@test@smallcaps`: Accommodation of the `\zf@thisfontfeature` change. 28

`\zf@update@ff`: Fix for featureless fonts (*e.g.*, the MS fonts) being ignored. 29

v1.8

`\AtBeginDocument`: Added support for user-specified `\mathrm` and others. 45

Finally fixed legacy maths font issues. Also checks that euler.sty is loaded in the right order. 45

`\setmathrm`: Implemented (with friends). 23

v1.8a

`\AtBeginDocument`: Added conditional to `\colon` math symbol (incompatibility with lucida and amsmath) 47

6 Implementation

Herein lies the implementation details of this package. Welcome! It's my first.

For some reason, I decided to prefix all the package internal command names and variables with `zf`. I don't know why I chose those letters, but I guess I just liked the look/feel of the letters together at the time.

```
⟨*fontspec⟩
```

Only proceed if it is XƎTEX that is doing the typesetting.

```
\expandafter\ifx\csname XeTeXversion\endcsname\relax
  \@latex@error{^^J
   ************************************************^^J
   *** Sorry!                                   ^^J
   *** The fontspec package only works with XeTeX.  ^^J
   ************************************************}
\fi
```

While the business of encoding names is up in the air, I'll continue to use the `U` encoding.

```
\def\zf@enc{U}
%\RequirePackage[\zf@enc]{fontenc}
\renewcommand\encodingdefault{\zf@enc}
```

`\define@key`
`\setkeys`

The keyval package makes processing comma separated inputs very simple with the `\define@key` and `\setkeys` commands:

- `\define@key{`⟨*family*⟩`}{`⟨*option*⟩`}{`⟨*commands to process*⟩`}`
- `\setkeys{`⟨*family*⟩`}{`⟨*input list*⟩`}`

`\define@key` is used to define each ⟨*option*⟩, grouped by ⟨*family*⟩, each of which executes its ⟨*commands to process*⟩ when it is included in an ⟨*input list*⟩ in a call to `\setkeys`.

```
\RequirePackage{keyval}
```

A typical example of what happens when this package runs various user commands is outlined below. I'll find some better way of slotting it into the real documentation sometime in the future.

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 family

```
WarnockPro+onum+dlig
```

(for use with `\fontfamily`, *etc.*)

4. which, in `\csname...\endcsname`, gives

   ```
   {Numbers=OldStyle,Ligatures=Rare}%
   {Warnock Pro}
   ```

5. This is used by

   ```
   \addfontfeature{Color=CC0000}
   ```

6. which is in this case equivalent to

   ```
   \fontspec[Numbers=OldStyle,%
             Ligatures=Rare,%
             Color=CC0000]{Warnock Pro}
   ```

7. and the process starts again.

6.1 User commands

This section contains the definitions of the commands detailed in the user documentation. Only the 'top level' definitions of the commands are contained herein; they all use or define macros which are defined or used later on in Section 6.2 on page 24.

6.1.1 Font selection

`\fontspec` This is the main command of the package that selects fonts with various features. It takes two arguments: the Mac OS X font name and the optional requested features of that font. It simply runs `\zf@fontspec`, which takes the same arguments as the top level macro and puts the new-fangled font family name into the global `\zf@family`. Then this new font family is selected.

```
\newcommand*\fontspec[2][]{%
  \zf@fontspec{#1}{#2}%
  \fontfamily\zf@family\selectfont}
```

`\setromanfont` `\setsansfont` `\setmonofont` The following three macros perform equivalent operations setting the default font (using `\let` rather than `\renewcommand` because `\zf@family` will change in the future) for a particular family: roman, sans serif, or typewriter (monospaced). I end them with `\normalfont` so that if they're used in the document, the change registers immediately.

```
\newcommand*\setromanfont[2][]{%
  \zf@fontspec{#1}{#2}%
  \let\rmdefault\zf@family
  \normalfont}
\newcommand*\setsansfont[2][]{%
  \zf@fontspec{#1}{#2}%
  \let\sfdefault\zf@family
  \normalfont}
\newcommand*\setmonofont[2][]{%
  \zf@fontspec{#1}{#2}%
  \let\ttdefault\zf@family
  \normalfont}
```

\setmathrm
\setmathsf
\setboldmathrm
\setmathtt

These commands are analogous to `\setromanfont` and others, but for selecting the font used for `\mathrm`, *etc*. They can only be used in the preamble of the document. `\setboldmathrm` is used for specifying which fonts should be used in `\boldmath`.

```
\newcommand*\setmathrm[2][]{%
  \zf@fontspec{#1}{#2}%
  \let\zf@rmmaths\zf@family}
\newcommand*\setboldmathrm[2][]{%
  \zf@fontspec{#1}{#2}%
  \let\zf@rmboldmaths\zf@family}
\newcommand*\setmathsf[2][]{%
  \zf@fontspec{#1}{#2}%
  \let\zf@sfmaths\zf@family}
\newcommand*\setmathtt[2][]{%
  \zf@fontspec{#1}{#2}%
  \let\zf@ttmaths\zf@family}
\@onlypreamble\setmathrm
\@onlypreamble\setboldmathrm
\@onlypreamble\setmathsf
\@onlypreamble\setmathtt
```

If the commands above are not executed, then `\rmdefault` (etc.) will be used.

```
\def\zf@rmmaths{\rmdefault}
\def\zf@sfmaths{\sfdefault}
\def\zf@ttmaths{\ttdefault}
```

\newfontinstance

This macro takes the arguments of `\fontspec` with a prepended ⟨*instance cmd*⟩ (code for middle optional argument generated by Scott Pakin's newcommand.py). This command is used when a specific font instance needs to be referred to repetitively (*e.g.*, in a section heading) since continuously calling `\zf@fontspec` is inefficient because it must parse the option arguments every time.

`\zf@fontspec` defines a font family and saves its name in `\zf@family`. So then this value is expanded in a temporary macro also containing the *unexpanded* commands to later select the font family to which it refers. Finally, the requested name for the font instance is `\let`ed from the temporary macro.

```
\newcommand*\newfontinstance[1]{%
  \@ifnextchar[{\newfontinstance@i#1}{\newfontinstance@i#1[]}}
\def\newfontinstance@i#1[#2]#3{%
  \zf@fontspec{#2}{#3}%
  \edef\zf@tempinst{\noexpand\fontfamily{\zf@family}\noexpand\selectfont}%
  \let#1\zf@tempinst}
```

6.1.2 Font feature selection

\defaultfontfeatures
\zf@default@options

This macro takes one argument that consists of all of feature options that will be applied by default to all subsequent `\fontspec`, et al., commands. It stores its value in `\zf@default@options` (initialised empty), which is concatenated with the individual macro choices in the `\zf@get@feature@requests` macro.

```
\newcommand*\defaultfontfeatures[1]{\def\zf@default@options{#1}}
\let\zf@default@options\@empty
```

\addfontfeatures In order to be able to extend the feature selection of a given font, two things need to be known: the currently selected features, and the currently selected font. Every time a font family is creating, this information is saved inside a control sequence with the name of the font family itself.

This macro extracts this information, then appends the requested font features to add to the already existing ones, and calls the font again with the top level `\fontspec` command.

The default options are *not* applied (which is why they're saved and restored with `\zf@default@options@old`), so this means that the only added features to the font are strictly those specified by this command.

`\addfontfeature` is defined as an alias, as I found that I often typed this instead when adding only a single font feature.

```
\newcommand*\addfontfeatures[1]{%
  \let\zf@default@options@old\zf@default@options
  \let\zf@default@options\@empty
  \edef\zf@thisinfo{\csname\f@family\endcsname}%
  \fontspec
    [\expandafter\@firstoftwo\zf@thisinfo, #1]%
    {\expandafter\@secondoftwo\zf@thisinfo}%
  \let\zf@default@options\zf@default@options@old}
\let\addfontfeature\addfontfeatures
```

6.1.3 Defining new font features

\newfontfeature `\newfontfeature` takes two arguments: the name of the feature tag by which to reference it, and the string that is used to select the font feature. It uses a counter to keep track of the number of new features introduced; every time a new feature is defined, a control sequence is defined made up of the concatenation of `+zf-` and the new feature tag. This long-winded control sequence is then called upon to update the font family string when a new instance is requested.

```
\newcounter{newff}
\newcommand*\newfontfeature[2]{%
  \stepcounter{newff}%
  \expandafter\edef\csname +zf-#1\endcsname{+zf-\thenewff}%
  \define@key{zf}{#1}[]{%
    \zf@update@family{\csname +zf-#1\endcsname}%
    \zf@update@ff{#2}}}
```

\newfeaturecode This is similar to above, but simply defines a new AAT feature code (second two arguments) with a new name (first argument). Maybe I should just give users access to the `\zf@make@feature` command?

```
\newcommand*\newfeaturecode[3]{%
  \define@key{zf}{#1}[]{\zf@make@feature{#2}{#3}{}}}
```

6.2 Internal macros

\zf@fontspec This is the command that defines font families for use. Given a list of font features (`#1`, stored in `\zf@options` for `\zf@get@feature@requests`) for a requested font (`#2`, stored in `\zf@macname` globally for the `\zf@set@font@type` and `\zf@make@aat@feature@string` macros), it will define an NFSS family for that font and put the family name into `\zf@family`.

Then we check with `\zf@set@font@type` whether the font is AAT or OpenType, and convert the requested features to font definition strings. This is performed with `\zf@get@feature@requests`, in which `\setkeys` retrieves the requested font features and processes them. To build up the complex family name, it concatenates each font feature with the family name of the font. So since `\setkeys` is run more than once (since different font faces may have different feature names), we only want the complex family name to be built up once, hence the `\zf@firsttime` conditionals.

```
\newcommand*\zf@fontspec[2]{%
  \edef\zf@options{#1}%
  \edef\zf@macname{#2}%
  \edef\zf@family{\zap@space #2 \@empty}%
  \zf@set@font@type{#2}%
  \zf@firsttimetrue
    \zf@get@feature@requests
  \zf@firsttimefalse
```

Now that we have the complex family name, we can check to see if the family has already been defined, and if not, do so. Once the family name is created, save *in* it the user string of the requested options and font name for use with `\addfont-features` (this info is separately braced for easy extraction with the `\@first-` and `\@secondoftwo` commands).

`\zf@current@options` is the option list prepended by the default option list, if any.

```
  \expandafter\ifx\csname\zf@family\endcsname\relax
    \wlog{fontspec: Defining font family for ” #2”
          with options [\zf@current@options]}%
    \edef\zf@info{{\zf@current@options}{#2}}%
    \expandafter\let\csname\zf@family\endcsname\zf@info
```

Next the font family and its shapes are defined in the NFSS.

All NFSS specifications take their default values, so if any of them are redefined, the shapes will be selected to fit in with the current state. For example, if `\bfdefault` is redefined to `b`, all bold shapes defined by this package will also be assigned to `b`.

The macros `\zf@bf`, et al., are used to store the name of the custom bold, et al., font, if requested as user options. If they are empty, the default fonts are used.

First we define the font family and define the normal shape: (the specified options are used implicitly)

```
    \DeclareFontFamily{\zf@enc}{\zf@family}{}%
    \zf@make@font@shapes{#2}{\mddefault}{\updefault}%
```

Secondly, bold. Start out by saving the current font features and appending to them, if any, the extra bold options defined with `BoldFeatures`. Then, the bold font is defined either as the ATS default (`\zf@make@font@shapes`' optional argument is to check if there actually is one; if not, the bold NFSS series is left undefined) or with the font specified with the `BoldFont` feature.

```
  \let\zf@options@old\zf@options
  \def\zf@options{\zf@options@old,\zf@bf@options}%
  \ifx\zf@bf\@empty
    \zf@make@font@shapes[#2]{#2/B}{\bfdefault}{\updefault}%
  \else
    \zf@make@font@shapes\zf@bf\bfdefault\updefault
  \fi
  \let\zf@options\zf@options@old
```

And italic in the same way:

```
  \let\zf@options@old\zf@options
  \def\zf@options{\zf@options@old,\zf@it@options}%
  \ifx\zf@it\@empty
    \zf@make@font@shapes[#2]{#2/I}{\mddefault}{\itdefault}%
  \else
    \zf@make@font@shapes\zf@it\mddefault\itdefault
  \fi
  \let\zf@options\zf@options@old
```

If requested, the custom fonts take precedence when choosing the bold italic font. When both italic and bold fonts are requested and the bold italic font hasn't been explicitly specified (a rare occurance, presumably), the new bold font is used to define the new bold italic font.

```
  \let\zf@options@old\zf@options
  \def\zf@options{\zf@options@old,\zf@bfit@options}%
  \ifx\zf@bfit\@empty
    \ifx\zf@bf\@empty
      \ifx\zf@it\@empty
        \zf@make@font@shapes[#2]{#2/BI}{\bfdefault}{\itdefault}%
      \else
        \zf@make@font@shapes[\zf@it]{\zf@it/B}{\bfdefault}{\itdefault}%
      \fi
    \else
      \zf@make@font@shapes[\zf@bf]{\zf@bf/I}{\bfdefault}{\itdefault}%
    \fi
  \else
    \zf@make@font@shapes\zf@bfit\bfdefault\itdefault
  \fi
  \let\zf@options\zf@options@old
\fi}
```

6.2.1 Fonts

`\zf@set@font@type` This macro sets `\zf@aat` or `\zf@opentype` booleans accordingly depending if the font in `\zf@macname` is an AAT font or an OpenType font, respectively.

```
\newcommand*\zf@set@font@type[1]{%
  \font\zf@testfont = "#1" at 10pt
  \zf@aatfalse \zf@opentypefalse \zf@mmfalse
  \expandafter\ifnum\XeTeXcountfeatures\zf@testfont > 0
    \zf@aattrue
  \fi
  \expandafter\ifnum\XeTeXOTcountscripts\zf@testfont > 0
```

```
    \zf@opentypetrue
  \fi
  \expandafter\ifnum\XeTeXcountvariations\zf@testfont > 0
    \zf@mmtrue
  \fi}
\newif\ifzf@aat \newif\ifzf@opentype \newif\ifzf@mm
```

`\zf@make@font@shapes` This macro uses `\DeclareFontShape` to define the font shape in question. It is necessary to wrap the whole thing in `\nfss@catcodes` so that the scale factor works. This necessitates `\globaldefs=1` so the macro is visible to the rest of the package. The mandatory arguments are: `#2` the font name, `#3` the font series, and `#4` the font shape. The optional argument is used when making the font shapes for bold, italic, and bold italic fonts using XƎTEX's auto-recognition with `/B`, `/I`, and `/BI` font name suffixes. If no such font is found, it falls back to the original font name, in which case this macro doesn't proceed.

```
\begingroup
  \nfss@catcodes
  \globaldefs=1
  \newcommand*\zf@make@font@shapes[4][]{%
    \ifEqFonts{#1}{#2}\then\else
      \edef\zf@macname{#2}%
      \zf@get@feature@requests
      \ifx\zf@scale\@empty
        \let\zf@scale@str\@empty
      \else
        \edef\zf@scale@str{s*[\zf@scale]}%
      \fi
      \DeclareFontShape{\zf@enc}{\zf@family}{#3}{#4}%
        {<-> \zf@scale@str " #2\zf@suffix:\zf@ff" }{}%
```

Next, the small caps are defined. `\zf@test@smallcaps` is used to define the appropriate string for activating small caps in the font, if they exist. If we are defining small caps for the upright shape, then the small caps shape default is used. For an *italic* font, however, the shape parameter is overloaded and we must call italic small caps by their own identifier. See Section 6.4 on page 45 for the code that enables this usage.

```
      \zf@test@smallcaps
      \ifx\zf@smallcaps\@empty\else
        \ifx #4\updefault
          \let\zf@scshape\scdefault
        \fi
        \ifx #4\itdefault
          \let\zf@scshape\sidefault
        \fi
        \DeclareFontShape
          {\zf@enc}{\zf@family}{#3}{\zf@scshape}%
          {<-> \zf@scale@str " #2\zf@suffix:\zf@ff\zf@smallcaps" }{}%
      \fi
    \fi}
\endgroup
```

\zf@update@family This macro is used to build up a complex family name based on its features.

\zf@firsttime is set true in \zf@fontspec only the first time \f@get@feature@requests is called, so that the family name is only created once.

```
\newcommand*{\zf@update@family}[1]{%
  \ifzf@firsttime
    \g@addto@macro\zf@family{#1}%
  \fi}
\newif\ifzf@firsttime
```

6.2.2 Features

\zf@get@feature@requests This macro is a wrapper for \setkeys which expands and adds a default specification to the original passed options. It begins by initialising the commands used to hold font-feature specific strings.

```
\newcommand*\zf@get@feature@requests{%
  \ifzf@firsttime
    \let\zf@scale       \@empty
    \let\zf@suffix      \@empty
    \let\zf@bf          \@empty
    \let\zf@it          \@empty
    \let\zf@bfit        \@empty
    \let\zf@bf@options  \@empty
    \let\zf@it@options  \@empty
    \let\zf@bfit@options\@empty
  \fi
  \let\zf@ff\@empty
  \ifx\zf@default@options\@empty
    \let\zf@current@options\zf@options
  \else
    \edef\zf@current@options{\zf@default@options,\zf@options}%
  \fi
  \edef\zf@process@options
    {\noexpand\setkeys{zf}{\zf@current@options}}%
  \zf@process@options}
```

\zf@test@smallcaps This macro checks if the font contains small caps, and if so creates the string for accessing them in \zf@smallcaps. The OpenType portion of this macro has been adapted / stolen from Jonathan Kew's osxfonts.sty package.

```
\newcommand*\zf@test@smallcaps{%
  \let\zf@smallcaps\@empty
  \ifzf@aat
    \zf@make@aat@feature@string{3}{3}%
    \ifx\zf@thisfontfeature\@empty\else
      \edef\zf@smallcaps{\zf@thisfontfeature;}%
    \fi
  \fi
  \ifzf@opentype
    \font\zf@testfont=" \zf@macname" at 10pt
    \count255 = \XeTeXOTcountscripts\zf@testfont
    \ifnum\count255 > 0
```

```
      \count255 = 0
      \zf@featurecount = \XeTeXOTcountfeatures\zf@testfont " 6C61746E " 0
      \loop\ifnum\count255 < \zf@featurecount
        \ifnum\XeTeXOTfeaturetag\zf@testfont " 6C61746E " 0
          \count255 = " 736D6370 % ' smcp'
          \edef\zf@smallcaps{+smcp,}%
          \count255 = \zf@featurecount
        \else
          \advance\count255 by 1
        \fi
      \repeat
    \fi
  \fi}
\newcount\zf@featurecount
```

`\zf@update@ff` `\zf@ff` is the string used to define the list of specific font features. Each time another font feature is requested, this macro is used to add that feature to the list. AAT features are separated by semicolons, OpenType features by commas.

```
\newcommand*\zf@update@ff[1]{%
  \begingroup
    \def\zf@feature@separator{;}%
    \ifzf@opentype
      \def\zf@feature@separator{,}%
    \fi
    \let\@tempa\zf@ff
    \xdef\zf@ff{\@tempa #1\zf@feature@separator}%
  \endgroup}
```

`\zf@make@feature` This macro is called by each feature key selected, and runs according to which type of font is selected. Note that it does not check to see if an OpenType feature exists.

```
\newcommand*\zf@make@feature[3]{%
  \ifzf@aat
    \zf@make@aat@feature@string{#1}{#2}%
    \ifx\zf@thisfontfeature\@empty\else
      \zf@update@family{+#1,#2}%
      \zf@update@ff\zf@thisfontfeature
    \fi
  \fi
  \ifzf@opentype
    \zf@update@family{#3}%
    \zf@update@ff{#3}%
  \fi}
```

`\zf@define@font@feature`
`\zf@define@feature@option` These macros are used in order to simplify font feature definition later on. `\aliasfeature` works for features defined with `\zf@define@font@feature`, but that's it for now. It remains as a reminder for me to get it working for all features.

```
\newcommand*\zf@define@font@feature[1]{%
  \define@key{zf}{#1}{{\setkeys{zf@feat@#1}{##1}}}}
\newcommand*\zf@define@feature@option[5]{%
```

```
  \define@key{zf@feat@#1}{#2}[]{\zf@make@feature{#3}{#4}{#5}}}
\newcommand*\aliasfeature[2]{%
  \define@key{zf}{#2}{{\setkeys{zf@feat@#1}{##1}}}}
```

\zf@make@aat@feature@string This macro takes the numerical codes for a font feature and creates a specified macro containing the string required in the font definition to turn that feature on or off. Used primarily in \zf@make@aat@feature, but also used to check if small caps exists in the requested font.

```
\newcommand*\zf@make@aat@feature@string[2]{%
  \font\zf@fontone=” \zf@macname”  at 10pt
  \edef\zf@this@featurename{\XeTeXfeaturename\zf@fontone #1}%
  \ifx\zf@this@featurename\@empty
    \let\zf@thisfontfeature\@empty
  \else
    \edef\zf@this@selectionname{\XeTeXselectorname\zf@fontone #1 #2}%
    \ifx\zf@this@selectionname\@empty
      \let\zf@thisfontfeature\@empty
    \else
      \edef\zf@thisfontfeature{%
        \ifnum\XeTeXisexclusivefeature\zf@fontone #1 > 0
          \zf@this@featurename=\zf@this@selectionname
        \else
          \ifodd #2
            \zf@this@featurename=!\zf@this@selectionname
          \else
            \zf@this@featurename=\zf@this@selectionname
          \fi
        \fi}%
    \fi
  \fi}
```

\ifEqStr \ifEqStr was a conditional for checking it two strings are the same. It no longer exists.

\ifEqFonts A conditional for checking if two fonts are the same. If one of the arguments is empty, special provision needs to be taken, since there is no font with an empty string for a name. This case arises when \zf@make@font@shapes doesn't take its optional argument.

```
\let\then\iftrue
\def\ifEqFonts#1#2\then{%
  \ifx#1\@empty\else
    \font\zf@fontone = ” #1”  at 10pt
    \edef\@tempa{\fontname\zf@fontone}%
  \fi
  \font\zf@fonttwo = ” #2”  at 10pt
  \edef\@tempb{\fontname\zf@fonttwo}%
  \ifx\@tempa\@tempb}
```

6.3 keyval definitions

This is the tedious section where we correlate all possible (eventually) font feature requests with their XeTeX representations.

6.3.1 Bold/italic choosing options

The `Bold`, `Italic`, and `BoldItalic` features are for defining explicitly the bold and italic fonts used in a font family. v1.6 introduced arbitrary font features for these shapes (`BoldFeatures`, etc.), so the names of the shape-selecting options were appended with `Font` for consistency.

```
\define@key{zf}{Bold}{%
  \def\zf@bf{#1}%
  \zf@update@family{bf:\zap@space #1 \@empty}}
\define@key{zf}{Italic}{%
  \def\zf@it{#1}%
  \zf@update@family{it:\zap@space #1 \@empty}}
\define@key{zf}{BoldItalic}{%
  \def\zf@bfit{#1}%
  \zf@update@family{bfit:\zap@space #1 \@empty}}
\define@key{zf}{BoldFont}{%
  \def\zf@bf{#1}%
  \zf@update@family{bf:\zap@space #1 \@empty}}
\define@key{zf}{ItalicFont}{%
  \def\zf@it{#1}%
  \zf@update@family{it:\zap@space #1 \@empty}}
\define@key{zf}{BoldItalicFont}{%
  \def\zf@bfit{#1}%
  \zf@update@family{bfit:\zap@space #1 \@empty}}

\define@key{zf}{BoldFeatures}{%
  \def\zf@bf@options{#1}%
  \zf@update@family{bffeat:\zap@space #1 \@empty}}
\define@key{zf}{ItalicFeatures}{%
  \def\zf@it@options{#1}%
  \zf@update@family{itfeat:\zap@space #1 \@empty}}
\define@key{zf}{BoldItalicFeatures}{%
  \def\zf@bfit@options{#1}%
  \zf@update@family{bfitfeat:\zap@space #1 \@empty}}
```

6.3.2 Font-independent options

These options can be applied to any font. Both spellings (American and correct) of 'colour' offered.

```
\define@key{zf}{Scale}{%
  \zf@update@family{+scale:#1}%
  \edef\zf@scale{#1}}
\define@key{zf}{Colour}{%
  \zf@update@family{+col:#1}%
  \zf@update@ff{color=#1}}
\define@key{zf}{Color}{%
```

```
  \zf@update@family{+col:#1}%
  \zf@update@ff{color=#1}}
\define@key{zf}{Mapping}{%
  \zf@update@family{+map:#1}%
  \zf@update@ff{mapping=#1}}
```

6.3.3 Continuous font axes

```
\define@key{zf}{Weight}{%
  \zf@update@family{+weight:#1}%
  \zf@update@ff{weight=#1}}
\define@key{zf}{Width}{%
  \zf@update@family{+width:#1}%
  \zf@update@ff{width=#1}}
\define@key{zf}{OpticalSize}{%
  \ifzf@opentype
    \g@addto@macro\zf@suffix{/S=#1}%
    \zf@update@family{+size:#1}
  \fi
  \ifzf@mm
    \zf@update@family{+size:#1}%
    \zf@update@ff{optical size=#1}
  \fi
  \ifzf@opentype\else
    \ifzf@mm\else
      \ifzf@firsttime
        \PackageWarning{fontspec}{\zf@macname doesn't appear to have an Opti-
cal Size axis}
      \fi
    \fi
  \fi}
```

6.3.4 OpenType script and language

```
\define@key{zf}{Script}{\ifzf@opentype\bgroup\setkeys{zf@scpt}{#1}\egroup\fi}
\newcommand*\zf@make@scpt@feature[2]{%
  \define@key{zf@scpt}{#1}[]{%
    \zf@update@family{+scpt:#2}%
    \zf@update@ff{script=#2}}}
\zf@make@scpt@feature{Arabic}{arab}
\zf@make@scpt@feature{Armenian}{armn}
\zf@make@scpt@feature{Balinese}{bali}
\zf@make@scpt@feature{Bengali}{beng}
\zf@make@scpt@feature{Bopomofo}{bopo}
\zf@make@scpt@feature{Braille}{brai}
\zf@make@scpt@feature{Buginese}{bugi}
\zf@make@scpt@feature{Buhid}{buhd}
\zf@make@scpt@feature{Byzantine Music}{byzm}
\zf@make@scpt@feature{Canadian Syllabics}{cans}
\zf@make@scpt@feature{Cherokee}{cher}
\zf@make@scpt@feature{CJK}{hani}
\zf@make@scpt@feature{CJK Ideographic}{hani}
\zf@make@scpt@feature{Coptic}{copt}
```

```
\zf@make@scpt@feature{Cypriot Syllabary}{cprt}
\zf@make@scpt@feature{Cyrillic}{cyrl}
\zf@make@scpt@feature{Default}{DFLT}
\zf@make@scpt@feature{Deseret}{dsrt}
\zf@make@scpt@feature{Devanagari}{deva}
\zf@make@scpt@feature{Ethiopic}{ethi}
\zf@make@scpt@feature{Georgian}{geor}
\zf@make@scpt@feature{Glagolitic}{glag}
\zf@make@scpt@feature{Gothic}{goth}
\zf@make@scpt@feature{Greek}{grek}
\zf@make@scpt@feature{Gujarati}{gujr}
\zf@make@scpt@feature{Gurmukhi}{guru}
\zf@make@scpt@feature{Hangul Jamo}{jamo}
\zf@make@scpt@feature{Hangul}{hang}
\zf@make@scpt@feature{Hanunoo}{hano}
\zf@make@scpt@feature{Hebrew}{hebr}
\zf@make@scpt@feature{Hiragana and Katakana}{kana}
\zf@make@scpt@feature{Kana}{kana}
\zf@make@scpt@feature{Javanese}{java}
\zf@make@scpt@feature{Kannada}{knda}
\zf@make@scpt@feature{Kharosthi}{khar}
\zf@make@scpt@feature{Khmer}{khmr}
\zf@make@scpt@feature{Lao}{lao }
\zf@make@scpt@feature{Latin}{latn}
\zf@make@scpt@feature{Limbu}{limb}
\zf@make@scpt@feature{Linear B}{linb}
\zf@make@scpt@feature{Malayalam}{mlym}
\zf@make@scpt@feature{Math}{math}
\zf@make@scpt@feature{Maths}{math}
\zf@make@scpt@feature{Mongolian}{mong}
\zf@make@scpt@feature{Musical Symbols}{musc}
\zf@make@scpt@feature{Myanmar}{mymr}
\zf@make@scpt@feature{N’ko}{nko }
\zf@make@scpt@feature{Ogham}{ogam}
\zf@make@scpt@feature{Old Italic}{ital}
\zf@make@scpt@feature{Old Persian Cuneiform}{xpeo}
\zf@make@scpt@feature{Oriya}{orya}
\zf@make@scpt@feature{Osmanya}{osma}
\zf@make@scpt@feature{Phags-pa}{phag}
\zf@make@scpt@feature{Phoenician}{phnx}
\zf@make@scpt@feature{Runic}{runr}
\zf@make@scpt@feature{Shavian}{shaw}
\zf@make@scpt@feature{Sinhala}{sinh}
\zf@make@scpt@feature{Sumero-Akkadian Cuneiform}{xsux}
\zf@make@scpt@feature{Syloti Nagri}{sylo}
\zf@make@scpt@feature{Syriac}{syrc}
\zf@make@scpt@feature{Tagalog}{tglg}
\zf@make@scpt@feature{Tagbanwa}{tagb}
\zf@make@scpt@feature{Tai Le}{tale}
\zf@make@scpt@feature{Tai Lu}{talu}
\zf@make@scpt@feature{Tamil}{taml}
```

```
\zf@make@scpt@feature{Telugu}{telu}
\zf@make@scpt@feature{Thaana}{thaa}
\zf@make@scpt@feature{Thai}{thai}
\zf@make@scpt@feature{Tibetan}{tibt}
\zf@make@scpt@feature{Tifinagh}{tfng}
\zf@make@scpt@feature{Ugaritic Cuneiform}{ugar}
\zf@make@scpt@feature{Yi}{yi  }

\define@key{zf}{Language}{\ifzf@opentype\bgroup\setkeys{zf@lang}{#1}\egroup\fi}
\newcommand*\zf@make@lang@feature[2]{%
  \define@key{zf@lang}{#1}[]{%
    \zf@update@family{+lang:#2}%
    \zf@update@ff{language=#2}}}
\zf@make@lang@feature{Abaza}{ABA}
\zf@make@lang@feature{Abkhazian}{ABK}
\zf@make@lang@feature{Adyghe}{ADY}
\zf@make@lang@feature{Afrikaans}{AFK}
\zf@make@lang@feature{Afar}{AFR}
\zf@make@lang@feature{Agaw}{AGW}
\zf@make@lang@feature{Altai}{ALT}
\zf@make@lang@feature{Amharic}{AMH}
\zf@make@lang@feature{Arabic}{ARA}
\zf@make@lang@feature{Aari}{ARI}
\zf@make@lang@feature{Arakanese}{ARK}
\zf@make@lang@feature{Assamese}{ASM}
\zf@make@lang@feature{Athapaskan}{ATH}
\zf@make@lang@feature{Avar}{AVR}
\zf@make@lang@feature{Awadhi}{AWA}
\zf@make@lang@feature{Aymara}{AYM}
\zf@make@lang@feature{Azeri}{AZE}
\zf@make@lang@feature{Badaga}{BAD}
\zf@make@lang@feature{Baghelkhandi}{BAG}
\zf@make@lang@feature{Balkar}{BAL}
\zf@make@lang@feature{Baule}{BAU}
\zf@make@lang@feature{Berber}{BBR}
\zf@make@lang@feature{Bench}{BCH}
\zf@make@lang@feature{Bible Cree}{BCR}
\zf@make@lang@feature{Belarussian}{BEL}
\zf@make@lang@feature{Bemba}{BEM}
\zf@make@lang@feature{Bengali}{BEN}
\zf@make@lang@feature{Bulgarian}{BGR}
\zf@make@lang@feature{Bhili}{BHI}
\zf@make@lang@feature{Bhojpuri}{BHO}
\zf@make@lang@feature{Bikol}{BIK}
\zf@make@lang@feature{Bilen}{BIL}
\zf@make@lang@feature{Blackfoot}{BKF}
\zf@make@lang@feature{Balochi}{BLI}
\zf@make@lang@feature{Balante}{BLN}
\zf@make@lang@feature{Balti}{BLT}
\zf@make@lang@feature{Bambara}{BMB}
\zf@make@lang@feature{Bamileke}{BML}
```

```
\zf@make@lang@feature{Breton}{BRE}
\zf@make@lang@feature{Brahui}{BRH}
\zf@make@lang@feature{Braj Bhasha}{BRI}
\zf@make@lang@feature{Burmese}{BRM}
\zf@make@lang@feature{Bashkir}{BSH}
\zf@make@lang@feature{Beti}{BTI}
\zf@make@lang@feature{Catalan}{CAT}
\zf@make@lang@feature{Cebuano}{CEB}
\zf@make@lang@feature{Chechen}{CHE}
\zf@make@lang@feature{Chaha Gurage}{CHG}
\zf@make@lang@feature{Chattisgarhi}{CHH}
\zf@make@lang@feature{Chichewa}{CHI}
\zf@make@lang@feature{Chukchi}{CHK}
\zf@make@lang@feature{Chipewyan}{CHP}
\zf@make@lang@feature{Cherokee}{CHR}
\zf@make@lang@feature{Chuvash}{CHU}
\zf@make@lang@feature{Comorian}{CMR}
\zf@make@lang@feature{Coptic}{COP}
\zf@make@lang@feature{Cree}{CRE}
\zf@make@lang@feature{Carrier}{CRR}
\zf@make@lang@feature{Crimean Tatar}{CRT}
\zf@make@lang@feature{Church Slavonic}{CSL}
\zf@make@lang@feature{Czech}{CSY}
\zf@make@lang@feature{Danish}{DAN}
\zf@make@lang@feature{Dargwa}{DAR}
\zf@make@lang@feature{Woods Cree}{DCR}
\zf@make@lang@feature{German}{DEU}
\zf@make@lang@feature{Default}{DFLT}
\zf@make@lang@feature{Dogri}{DGR}
\zf@make@lang@feature{Divehi}{DIV}
\zf@make@lang@feature{Djerma}{DJR}
\zf@make@lang@feature{Dangme}{DNG}
\zf@make@lang@feature{Dinka}{DNK}
\zf@make@lang@feature{Dungan}{DUN}
\zf@make@lang@feature{Dzongkha}{DZN}
\zf@make@lang@feature{Ebira}{EBI}
\zf@make@lang@feature{Eastern Cree}{ECR}
\zf@make@lang@feature{Edo}{EDO}
\zf@make@lang@feature{Efik}{EFI}
\zf@make@lang@feature{Greek}{ELL}
\zf@make@lang@feature{English}{ENG}
\zf@make@lang@feature{Erzya}{ERZ}
\zf@make@lang@feature{Spanish}{ESP}
\zf@make@lang@feature{Estonian}{ETI}
\zf@make@lang@feature{Basque}{EUQ}
\zf@make@lang@feature{Evenki}{EVK}
\zf@make@lang@feature{Even}{EVN}
\zf@make@lang@feature{Ewe}{EWE}
\zf@make@lang@feature{French Antillean}{FAN}
\zf@make@lang@feature{Farsi}{FAR}
\zf@make@lang@feature{Finnish}{FIN}
```

```
\zf@make@lang@feature{Fijian}{FJI}
\zf@make@lang@feature{Flemish}{FLE}
\zf@make@lang@feature{Forest Nenets}{FNE}
\zf@make@lang@feature{Fon}{FON}
\zf@make@lang@feature{Faroese}{FOS}
\zf@make@lang@feature{French}{FRA}
\zf@make@lang@feature{Frisian}{FRI}
\zf@make@lang@feature{Friulian}{FRL}
\zf@make@lang@feature{Futa}{FTA}
\zf@make@lang@feature{Fulani}{FUL}
\zf@make@lang@feature{Ga}{GAD}
\zf@make@lang@feature{Gaelic}{GAE}
\zf@make@lang@feature{Gagauz}{GAG}
\zf@make@lang@feature{Galician}{GAL}
\zf@make@lang@feature{Garshuni}{GAR}
\zf@make@lang@feature{Garhwali}{GAW}
\zf@make@lang@feature{Ge'ez}{GEZ}
\zf@make@lang@feature{Gilyak}{GIL}
\zf@make@lang@feature{Gumuz}{GMZ}
\zf@make@lang@feature{Gondi}{GON}
\zf@make@lang@feature{Greenlandic}{GRN}
\zf@make@lang@feature{Garo}{GRO}
\zf@make@lang@feature{Guarani}{GUA}
\zf@make@lang@feature{Gujarati}{GUJ}
\zf@make@lang@feature{Haitian}{HAI}
\zf@make@lang@feature{Halam}{HAL}
\zf@make@lang@feature{Harauti}{HAR}
\zf@make@lang@feature{Hausa}{HAU}
\zf@make@lang@feature{Hawaiin}{HAW}
\zf@make@lang@feature{Hammer-Banna}{HBN}
\zf@make@lang@feature{Hiligaynon}{HIL}
\zf@make@lang@feature{Hindi}{HIN}
\zf@make@lang@feature{High Mari}{HMA}
\zf@make@lang@feature{Hindko}{HND}
\zf@make@lang@feature{Ho}{HO}
\zf@make@lang@feature{Harari}{HRI}
\zf@make@lang@feature{Croatian}{HRV}
\zf@make@lang@feature{Hungarian}{HUN}
\zf@make@lang@feature{Armenian}{HYE}
\zf@make@lang@feature{Igbo}{IBO}
\zf@make@lang@feature{Ijo}{IJO}
\zf@make@lang@feature{Ilokano}{ILO}
\zf@make@lang@feature{Indonesian}{IND}
\zf@make@lang@feature{Ingush}{ING}
\zf@make@lang@feature{Inuktitut}{INU}
\zf@make@lang@feature{Irish}{IRI}
\zf@make@lang@feature{Irish Traditional}{IRT}
\zf@make@lang@feature{Icelandic}{ISL}
\zf@make@lang@feature{Inari Sami}{ISM}
\zf@make@lang@feature{Italian}{ITA}
\zf@make@lang@feature{Hebrew}{IWR}
```

```
\zf@make@lang@feature{Javanese}{JAV}
\zf@make@lang@feature{Yiddish}{JII}
\zf@make@lang@feature{Japanese}{JAN}
\zf@make@lang@feature{Judezmo}{JUD}
\zf@make@lang@feature{Jula}{JUL}
\zf@make@lang@feature{Kabardian}{KAB}
\zf@make@lang@feature{Kachchi}{KAC}
\zf@make@lang@feature{Kalenjin}{KAL}
\zf@make@lang@feature{Kannada}{KAN}
\zf@make@lang@feature{Karachay}{KAR}
\zf@make@lang@feature{Georgian}{KAT}
\zf@make@lang@feature{Kazakh}{KAZ}
\zf@make@lang@feature{Kebena}{KEB}
\zf@make@lang@feature{Khutsuri Georgian}{KGE}
\zf@make@lang@feature{Khakass}{KHA}
\zf@make@lang@feature{Khanty-Kazim}{KHK}
\zf@make@lang@feature{Khmer}{KHM}
\zf@make@lang@feature{Khanty-Shurishkar}{KHS}
\zf@make@lang@feature{Khanty-Vakhi}{KHV}
\zf@make@lang@feature{Khowar}{KHW}
\zf@make@lang@feature{Kikuyu}{KIK}
\zf@make@lang@feature{Kirghiz}{KIR}
\zf@make@lang@feature{Kisii}{KIS}
\zf@make@lang@feature{Kokni}{KKN}
\zf@make@lang@feature{Kalmyk}{KLM}
\zf@make@lang@feature{Kamba}{KMB}
\zf@make@lang@feature{Kumaoni}{KMN}
\zf@make@lang@feature{Komo}{KMO}
\zf@make@lang@feature{Komso}{KMS}
\zf@make@lang@feature{Kanuri}{KNR}
\zf@make@lang@feature{Kodagu}{KOD}
\zf@make@lang@feature{Korean Old Hangul}{KOH}
\zf@make@lang@feature{Konkani}{KOK}
\zf@make@lang@feature{Kikongo}{KON}
\zf@make@lang@feature{Komi-Permyak}{KOP}
\zf@make@lang@feature{Korean}{KOR}
\zf@make@lang@feature{Komi-Zyrian}{KOZ}
\zf@make@lang@feature{Kpelle}{KPL}
\zf@make@lang@feature{Krio}{KRI}
\zf@make@lang@feature{Karakalpak}{KRK}
\zf@make@lang@feature{Karelian}{KRL}
\zf@make@lang@feature{Karaim}{KRM}
\zf@make@lang@feature{Karen}{KRN}
\zf@make@lang@feature{Koorete}{KRT}
\zf@make@lang@feature{Kashmiri}{KSH}
\zf@make@lang@feature{Khasi}{KSI}
\zf@make@lang@feature{Kildin Sami}{KSM}
\zf@make@lang@feature{Kui}{KUI}
\zf@make@lang@feature{Kulvi}{KUL}
\zf@make@lang@feature{Kumyk}{KUM}
\zf@make@lang@feature{Kurdish}{KUR}
```

```
\zf@make@lang@feature{Kurukh}{KUU}
\zf@make@lang@feature{Kuy}{KUY}
\zf@make@lang@feature{Koryak}{KYK}
\zf@make@lang@feature{Ladin}{LAD}
\zf@make@lang@feature{Lahuli}{LAH}
\zf@make@lang@feature{Lak}{LAK}
\zf@make@lang@feature{Lambani}{LAM}
\zf@make@lang@feature{Lao}{LAO}
\zf@make@lang@feature{Latin}{LAT}
\zf@make@lang@feature{Laz}{LAZ}
\zf@make@lang@feature{L-Cree}{LCR}
\zf@make@lang@feature{Ladakhi}{LDK}
\zf@make@lang@feature{Lezgi}{LEZ}
\zf@make@lang@feature{Lingala}{LIN}
\zf@make@lang@feature{Low Mari}{LMA}
\zf@make@lang@feature{Limbu}{LMB}
\zf@make@lang@feature{Lomwe}{LMW}
\zf@make@lang@feature{Lower Sorbian}{LSB}
\zf@make@lang@feature{Lule Sami}{LSM}
\zf@make@lang@feature{Lithuanian}{LTH}
\zf@make@lang@feature{Luba}{LUB}
\zf@make@lang@feature{Luganda}{LUG}
\zf@make@lang@feature{Luhya}{LUH}
\zf@make@lang@feature{Luo}{LUO}
\zf@make@lang@feature{Latvian}{LVI}
\zf@make@lang@feature{Majang}{MAJ}
\zf@make@lang@feature{Makua}{MAK}
\zf@make@lang@feature{Malayalam Traditional}{MAL}
\zf@make@lang@feature{Mansi}{MAN}
\zf@make@lang@feature{Marathi}{MAR}
\zf@make@lang@feature{Marwari}{MAW}
\zf@make@lang@feature{Mbundu}{MBN}
\zf@make@lang@feature{Manchu}{MCH}
\zf@make@lang@feature{Moose Cree}{MCR}
\zf@make@lang@feature{Mende}{MDE}
\zf@make@lang@feature{Me’en}{MEN}
\zf@make@lang@feature{Mizo}{MIZ}
\zf@make@lang@feature{Macedonian}{MKD}
\zf@make@lang@feature{Male}{MLE}
\zf@make@lang@feature{Malagasy}{MLG}
\zf@make@lang@feature{Malinke}{MLN}
\zf@make@lang@feature{Malayalam Reformed}{MLR}
\zf@make@lang@feature{Malay}{MLY}
\zf@make@lang@feature{Mandinka}{MND}
\zf@make@lang@feature{Mongolian}{MNG}
\zf@make@lang@feature{Manipuri}{MNI}
\zf@make@lang@feature{Maninka}{MNK}
\zf@make@lang@feature{Manx Gaelic}{MNX}
\zf@make@lang@feature{Moksha}{MOK}
\zf@make@lang@feature{Moldavian}{MOL}
\zf@make@lang@feature{Mon}{MON}
```

```
\zf@make@lang@feature{Moroccan}{MOR}
\zf@make@lang@feature{Maori}{MRI}
\zf@make@lang@feature{Maithili}{MTH}
\zf@make@lang@feature{Maltese}{MTS}
\zf@make@lang@feature{Mundari}{MUN}
\zf@make@lang@feature{Naga-Assamese}{NAG}
\zf@make@lang@feature{Nanai}{NAN}
\zf@make@lang@feature{Naskapi}{NAS}
\zf@make@lang@feature{N-Cree}{NCR}
\zf@make@lang@feature{Ndebele}{NDB}
\zf@make@lang@feature{Ndonga}{NDG}
\zf@make@lang@feature{Nepali}{NEP}
\zf@make@lang@feature{Newari}{NEW}
\zf@make@lang@feature{Nagari}{NGR}
\zf@make@lang@feature{Norway House Cree}{NHC}
\zf@make@lang@feature{Nisi}{NIS}
\zf@make@lang@feature{Niuean}{NIU}
\zf@make@lang@feature{Nkole}{NKL}
\zf@make@lang@feature{N’ko}{NKO}
\zf@make@lang@feature{Dutch}{NLD}
\zf@make@lang@feature{Nogai}{NOG}
\zf@make@lang@feature{Norwegian}{NOR}
\zf@make@lang@feature{Northern Sami}{NSM}
\zf@make@lang@feature{Northern Tai}{NTA}
\zf@make@lang@feature{Esperanto}{NTO}
\zf@make@lang@feature{Nynorsk}{NYN}
\zf@make@lang@feature{Oji-Cree}{OCR}
\zf@make@lang@feature{Ojibway}{OJB}
\zf@make@lang@feature{Oriya}{ORI}
\zf@make@lang@feature{Oromo}{ORO}
\zf@make@lang@feature{Ossetian}{OSS}
\zf@make@lang@feature{Palestinian Aramaic}{PAA}
\zf@make@lang@feature{Pali}{PAL}
\zf@make@lang@feature{Punjabi}{PAN}
\zf@make@lang@feature{Palpa}{PAP}
\zf@make@lang@feature{Pashto}{PAS}
\zf@make@lang@feature{Polytonic Greek}{PGR}
\zf@make@lang@feature{Pilipino}{PIL}
\zf@make@lang@feature{Palaung}{PLG}
\zf@make@lang@feature{Polish}{PLK}
\zf@make@lang@feature{Provencal}{PRO}
\zf@make@lang@feature{Portuguese}{PTG}
\zf@make@lang@feature{Chin}{QIN}
\zf@make@lang@feature{Rajasthani}{RAJ}
\zf@make@lang@feature{R-Cree}{RCR}
\zf@make@lang@feature{Russian Buriat}{RBU}
\zf@make@lang@feature{Riang}{RIA}
\zf@make@lang@feature{Rhaeto-Romanic}{RMS}
\zf@make@lang@feature{Romanian}{ROM}
\zf@make@lang@feature{Romany}{ROY}
\zf@make@lang@feature{Rusyn}{RSY}
```

```
\zf@make@lang@feature{Ruanda}{RUA}
\zf@make@lang@feature{Russian}{RUS}
\zf@make@lang@feature{Sadri}{SAD}
\zf@make@lang@feature{Sanskrit}{SAN}
\zf@make@lang@feature{Santali}{SAT}
\zf@make@lang@feature{Sayisi}{SAY}
\zf@make@lang@feature{Sekota}{SEK}
\zf@make@lang@feature{Selkup}{SEL}
\zf@make@lang@feature{Sango}{SGO}
\zf@make@lang@feature{Shan}{SHN}
\zf@make@lang@feature{Sibe}{SIB}
\zf@make@lang@feature{Sidamo}{SID}
\zf@make@lang@feature{Silte Gurage}{SIG}
\zf@make@lang@feature{Skolt Sami}{SKS}
\zf@make@lang@feature{Slovak}{SKY}
\zf@make@lang@feature{Slavey}{SLA}
\zf@make@lang@feature{Slovenian}{SLV}
\zf@make@lang@feature{Somali}{SML}
\zf@make@lang@feature{Samoan}{SMO}
\zf@make@lang@feature{Sena}{SNA}
\zf@make@lang@feature{Sindhi}{SND}
\zf@make@lang@feature{Sinhalese}{SNH}
\zf@make@lang@feature{Soninke}{SNK}
\zf@make@lang@feature{Sodo Gurage}{SOG}
\zf@make@lang@feature{Sotho}{SOT}
\zf@make@lang@feature{Albanian}{SQI}
\zf@make@lang@feature{Serbian}{SRB}
\zf@make@lang@feature{Saraiki}{SRK}
\zf@make@lang@feature{Serer}{SRR}
\zf@make@lang@feature{South Slavey}{SSL}
\zf@make@lang@feature{Southern Sami}{SSM}
\zf@make@lang@feature{Suri}{SUR}
\zf@make@lang@feature{Svan}{SVA}
\zf@make@lang@feature{Swedish}{SVE}
\zf@make@lang@feature{Swadaya Aramaic}{SWA}
\zf@make@lang@feature{Swahili}{SWK}
\zf@make@lang@feature{Swazi}{SWZ}
\zf@make@lang@feature{Sutu}{SXT}
\zf@make@lang@feature{Syriac}{SYR}
\zf@make@lang@feature{Tabasaran}{TAB}
\zf@make@lang@feature{Tajiki}{TAJ}
\zf@make@lang@feature{Tamil}{TAM}
\zf@make@lang@feature{Tatar}{TAT}
\zf@make@lang@feature{TH-Cree}{TCR}
\zf@make@lang@feature{Telugu}{TEL}
\zf@make@lang@feature{Tongan}{TGN}
\zf@make@lang@feature{Tigre}{TGR}
\zf@make@lang@feature{Tigrinya}{TGY}
\zf@make@lang@feature{Thai}{THA}
\zf@make@lang@feature{Tahitian}{THT}
\zf@make@lang@feature{Tibetan}{TIB}
```

```
\zf@make@lang@feature{Turkmen}{TKM}
\zf@make@lang@feature{Temne}{TMN}
\zf@make@lang@feature{Tswana}{TNA}
\zf@make@lang@feature{Tundra Nenets}{TNE}
\zf@make@lang@feature{Tonga}{TNG}
\zf@make@lang@feature{Todo}{TOD}
\zf@make@lang@feature{Turkish}{TRK}
\zf@make@lang@feature{Tsonga}{TSG}
\zf@make@lang@feature{Turoyo Aramaic}{TUA}
\zf@make@lang@feature{Tulu}{TUL}
\zf@make@lang@feature{Tuvin}{TUV}
\zf@make@lang@feature{Twi}{TWI}
\zf@make@lang@feature{Udmurt}{UDM}
\zf@make@lang@feature{Ukrainian}{UKR}
\zf@make@lang@feature{Urdu}{URD}
\zf@make@lang@feature{Upper Sorbian}{USB}
\zf@make@lang@feature{Uyghur}{UYG}
\zf@make@lang@feature{Uzbek}{UZB}
\zf@make@lang@feature{Venda}{VEN}
\zf@make@lang@feature{Vietnamese}{VIT}
\zf@make@lang@feature{Wa}{WA}
\zf@make@lang@feature{Wagdi}{WAG}
\zf@make@lang@feature{West-Cree}{WCR}
\zf@make@lang@feature{Welsh}{WEL}
\zf@make@lang@feature{Wolof}{WLF}
\zf@make@lang@feature{Tai Lue}{XBD}
\zf@make@lang@feature{Xhosa}{XHS}
\zf@make@lang@feature{Yakut}{YAK}
\zf@make@lang@feature{Yoruba}{YBA}
\zf@make@lang@feature{Y-Cree}{YCR}
\zf@make@lang@feature{Yi Classic}{YIC}
\zf@make@lang@feature{Yi Modern}{YIM}
\zf@make@lang@feature{Chinese Hong Kong}{ZHH}
\zf@make@lang@feature{Chinese Phonetic}{ZHP}
\zf@make@lang@feature{Chinese Simplified}{ZHS}
\zf@make@lang@feature{Chinese Traditional}{ZHT}
\zf@make@lang@feature{Zande}{ZND}
\zf@make@lang@feature{Zulu}{ZUL}
```

6.3.5 Ligatures

The call to the nested keyval family must be wrapped in braces to hide the parent list (this later requires the use of global definitions (`\xdef`) in `\zf@update@`...). Both AAT and OpenType names are offered to chose `Rare`/`Discretionary` ligatures.

```
\zf@define@font@feature{Ligatures}
\zf@define@feature@option{Ligatures}{Required}{1}{0}{+rlig}
\zf@define@feature@option{Ligatures}{NoRequired}     {1}{1} {-rlig}
\zf@define@feature@option{Ligatures}{Common}         {1}{2} {+liga}
\zf@define@feature@option{Ligatures}{NoCommon}       {1}{3} {-liga}
\zf@define@feature@option{Ligatures}{Rare}           {1}{4} {+dlig}
\zf@define@feature@option{Ligatures}{NoRare}         {1}{5} {-dlig}
\zf@define@feature@option{Ligatures}{Discretionary}  {1}{4} {+dlig}
```

```
\zf@define@feature@option{Ligatures}{NoDiscretionary}{1}{5} {-dlig}
\zf@define@feature@option{Ligatures}{Contextual}     {}{}   {+clig}
\zf@define@feature@option{Ligatures}{NoContextual}   {}{}   {-clig}
\zf@define@feature@option{Ligatures}{Historical}     {}{}   {+hlig}
\zf@define@feature@option{Ligatures}{NoHistorical}   {}{}   {-hlig}
\zf@define@feature@option{Ligatures}{Logos}          {1}{6} {}
\zf@define@feature@option{Ligatures}{NoLogos}        {1}{7} {}
\zf@define@feature@option{Ligatures}{Rebus}          {1}{8} {}
\zf@define@feature@option{Ligatures}{NoRebus}        {1}{9} {}
\zf@define@feature@option{Ligatures}{Diphthong}      {1}{10}{}
\zf@define@feature@option{Ligatures}{NoDiphthong}    {1}{11}{}
\zf@define@feature@option{Ligatures}{Squared}        {1}{12}{}
\zf@define@feature@option{Ligatures}{NoSquared}      {1}{13}{}
\zf@define@feature@option{Ligatures}{AbbrevSquared}  {1}{14}{}
\zf@define@feature@option{Ligatures}{NoAbbrevSquared}{1}{15}{}
\zf@define@feature@option{Ligatures}{Icelandic}      {1}{32}{}
\zf@define@feature@option{Ligatures}{NoIcelandic}    {1}{33}{}
```

6.3.6 Letters

```
\zf@define@font@feature{Letters}
\zf@define@feature@option{Letters}{Normal}{3}{0}{}
\zf@define@feature@option{Letters}{Uppercase}{3}{1}{+cpsp}
\zf@define@feature@option{Letters}{Lowercase}{3}{2}{}
\zf@define@feature@option{Letters}{SmallCaps}{3}{3}{+smcp}
\zf@define@feature@option{Letters}{PetiteCaps}{}{}{+pcap}
\zf@define@feature@option{Letters}{SMALLCAPS}{}{}{+c2sc}
\zf@define@feature@option{Letters}{PETITECAPS}{}{}{+c2pc}
\zf@define@feature@option{Letters}{InitialCaps}{3}{4}{}
\zf@define@feature@option{Letters}{Unicase}{}{}{+unic}
```

6.3.7 Numbers

These were originally separated into `NumberCase` and `NumberSpacing` following AAT, but it makes more sense to combine them.

Both naming conventions are offered to select the number case.

```
\zf@define@font@feature{Numbers}
\zf@define@feature@option{Numbers}{Monospaced}{6}{0}{+tnum}
\zf@define@feature@option{Numbers}{Proportional}{6}{1}{+pnum}
\zf@define@feature@option{Numbers}{Lowercase}{21}{0}{+onum}
\zf@define@feature@option{Numbers}{OldStyle}{21}{0}{+onum}
\zf@define@feature@option{Numbers}{Uppercase}{21}{1}{+lnum}
\zf@define@feature@option{Numbers}{Lining}{21}{1}{+lnum}
\zf@define@feature@option{Numbers}{SlashedZero}{14}{5}{+zero}
\zf@define@feature@option{Numbers}{NoSlashedZero}{14}{4}{-zero}
```

6.3.8 Swashes

```
\zf@define@font@feature{Swashes}
\zf@define@feature@option{Swashes}{Contextual}{}{}{+cswh}
\zf@define@feature@option{Swashes}{WordInitial}{8}{0}{+init}
\zf@define@feature@option{Swashes}{NoWordInitial}{8}{1}{-init}
```

```
\zf@define@feature@option{Swashes}{WordFinal}{8}{2}{+fina}
\zf@define@feature@option{Swashes}{NoWordFinal}{8}{3}{-fina}
\zf@define@feature@option{Swashes}{LineInitial}{8}{4}{}
\zf@define@feature@option{Swashes}{NoLineInitial}{8}{5}{}
\zf@define@feature@option{Swashes}{LineFinal}{8}{6}{+falt}
\zf@define@feature@option{Swashes}{NoLineFinal}{8}{7}{-falt}
\zf@define@feature@option{Swashes}{Inner}{8}{8}{}
\zf@define@feature@option{Swashes}{NoInner}{8}{9}{}
```

6.3.9 Diacritics

```
\zf@define@font@feature{Diacritics}
\zf@define@feature@option{Diacritics}{Show}{9}{0}{}
\zf@define@feature@option{Diacritics}{Hide}{9}{1}{}
\zf@define@feature@option{Diacritics}{Decompose}{9}{2}{}
```

6.3.10 Vertical position

```
\zf@define@font@feature{VerticalPosition}
\zf@define@feature@option{VerticalPosition}{Normal}{10}{0}{}
\zf@define@feature@option{VerticalPosition}{Superior}{10}{1}{+sups}
\zf@define@feature@option{VerticalPosition}{Inferior}{10}{2}{+subs}
\zf@define@feature@option{VerticalPosition}{ScientificInferior}{}{}{+sinf}
\zf@define@feature@option{VerticalPosition}{Ordinal}{10}{3}{+ordn}
\zf@define@feature@option{VerticalPosition}{Numerator}{}{}{+numr}
\zf@define@feature@option{VerticalPosition}{Denominator}{}{}{+dnom}
```

6.3.11 Fractions

```
\zf@define@font@feature{Fractions}
\zf@define@feature@option{Fractions}{On}{11}{1}{+frac}
\zf@define@feature@option{Fractions}{Off}{11}{0}{-frac}
\zf@define@feature@option{Fractions}{Diagonal}{11}{2}{}
\zf@define@feature@option{Fractions}{Alternate}{}{}{+afrc}
```

6.3.12 Alternates (AAT) and variants

Selected numerically because they don't have standard names. Very easy to process, very annoying for the user!

```
\define@key{zf}{Alternate}{\zf@make@feature{17}{#1}{}}
\define@key{zf}{Variant}{\zf@make@feature{18}{#1}{%
  \ifnum #1 > 0
    \ifnum #1 < 100
      \ifnum #1 > 9
        +ss#1%
      \else
        +ss0#1%
      \fi
    \else
      \PackageWarning{fontspec}{OpenType variant number must be
        between 1 and 99 inclusive.}%
    \fi
  \else
    \PackageWarning{fontspec}{OpenType variant number must be between 1
      and 99 inclusive.}%
  \fi}}
```

6.3.13 Style options

```
\zf@define@font@feature{Style}
\aliasfeature{Style}{StyleOptions}
\zf@define@feature@option{Style}{Alternate}{}{}{+salt}
\zf@define@feature@option{Style}{Italic}{32}{2}{+ital}
\zf@define@feature@option{Style}{Ruby}{28}{2}{+ruby}
\zf@define@feature@option{Style}{Swash}{}{}{+swsh}
\zf@define@feature@option{Style}{Display}{19}{1}{}
\zf@define@feature@option{Style}{Engraved}{19}{2}{}
\zf@define@feature@option{Style}{TitlingCaps}{19}{4}{+titl}
\zf@define@feature@option{Style}{TallCaps}{19}{5}{}
```

6.3.14 Character Shape

```
\zf@define@font@feature{CharacterShape}
\zf@define@feature@option{CharacterShape}{Traditional}{20}{0}{+trad}
\zf@define@feature@option{CharacterShape}{Simplified}{20}{1}{+smpl}
\zf@define@feature@option{CharacterShape}{JIS1978}{20}{2}{+jp78}
\zf@define@feature@option{CharacterShape}{JIS1983}{20}{3}{+jp83}
\zf@define@feature@option{CharacterShape}{JIS1990}{20}{4}{+jp90}
\zf@define@feature@option{CharacterShape}{Expert}{20}{10}{+expt}
\zf@define@feature@option{CharacterShape}{NLC}{20}{13}{+nlck}
```

6.3.15 Text Spacing

```
\zf@define@font@feature{TextSpacing}
\zf@define@feature@option{TextSpacing}{Proportional}{22}{0}{+pwid}
\zf@define@feature@option{TextSpacing}{FullWidth}{22}{1}{+fwid}
\zf@define@feature@option{TextSpacing}{HalfWidth}{22}{2}{+hwid}
\zf@define@feature@option{TextSpacing}{ThirdWidth}{22}{3}{+twid}
\zf@define@feature@option{TextSpacing}{QuarterWidth}{22}{4}{+qwid}
\zf@define@feature@option{TextSpacing}{AlternateProportional}{22}{5}{+palt}
\zf@define@feature@option{TextSpacing}{AlternateHalfWidth}{22}{6}{+halt}
\zf@define@feature@option{TextSpacing}{Default}{22}{7}{}
```

6.3.16 Annotation

```
\zf@define@font@feature{Annotation}
\zf@define@feature@option{Annotation}{Off}{24}{0}{-nalt}
\zf@define@feature@option{Annotation}{On}{}{}{+nalt}
\zf@define@feature@option{Annotation}{Box}{24}{1}{}
\zf@define@feature@option{Annotation}{RoundedBox}{24}{2}{}
\zf@define@feature@option{Annotation}{Circle}{24}{3}{}
\zf@define@feature@option{Annotation}{BlackCircle}{24}{4}{}
\zf@define@feature@option{Annotation}{Parenthesis}{24}{5}{}
\zf@define@feature@option{Annotation}{Period}{24}{6}{}
\zf@define@feature@option{Annotation}{RomanNumerals}{24}{7}{}
\zf@define@feature@option{Annotation}{Diamond}{24}{8}{}
\zf@define@feature@option{Annotation}{BlackSquare}{24}{9}{}
\zf@define@feature@option{Annotation}{BlackRoundSquare}{24}{10}{}
\zf@define@feature@option{Annotation}{DoubleCircle}{24}{11}{}
```

6.4 Italic small caps

The following code for utilising italic small caps sensibly is taken pretty much verbatim from Philip Lehman's *Font Installation Guide*. Note that `\upshape` needs to be used *twice* to get from italic small caps to regular upright (it always goes to small caps, then regular upright).

`\sishape` `\textsi` First, the commands for actually selecting italic small caps are defined. I use `si` as the NFSS shape for italic small caps, but I have seen `itsc` and `slsc` also used. `\sidefault` may be redefined to one of these if required for compatibility.

```
\providecommand*{\sidefault}{si}
\DeclareRobustCommand{\sishape}{%
  \not@math@alphabet\sishape\relax
  \fontshape\sidefault\selectfont}
\DeclareTextFontCommand{\textsi}{\sishape}
```

`\zf@merge@shape` This is the macro which enables the overload on the `\..shape` commands. It takes three such arguments. In essence, the macro selects the first argument, unless the second argument is already selected, in which case it selects the third.

```
\newcommand*{\zf@merge@shape}[3]{%
  \edef\@tempa{#1}%
  \edef\@tempb{#2}%
  \ifx\f@shape\@tempb
    \expandafter\ifx
    \csname\f@encoding/\f@family/\f@series/#3\endcsname
    \relax\else
      \edef\@tempa{#3}%
    \fi
  \fi
  \fontshape{\@tempa}\selectfont}
```

`\itshape` `\scshape` `\upshape` Here the original `\..shape` commands are redefined to use the merge shape macro.

```
\DeclareRobustCommand{\itshape}{%
  \not@math@alphabet\itshape\mathit
  \zf@merge@shape\itdefault\scdefault\sidefault}
\DeclareRobustCommand{\scshape}{%
  \not@math@alphabet\scshape\relax
  \zf@merge@shape\scdefault\itdefault\sidefault}
\DeclareRobustCommand{\upshape}{%
  \not@math@alphabet\upshape\relax
  \zf@merge@shape\updefault\sidefault\scdefault}
```

6.5 Selecting maths fonts

Here, the fonts used in math mode are redefined to correspond to the default roman, sans serif and typewriter fonts. Unfortunately, you can only define maths fonts in the preamble, otherwise I'd run this code whenever `\setromanfont` and friends was run.

`\AtBeginDocument` Everything here is performed `\AtBeginDocument` in order to overwrite euler's attempt. This means fontspec must be loaded *before* euler. We set up a conditional to return an error if this rule is violated.

Since every maths setup is slightly different, we also take different paths for defining various math glyphs depending which maths font package has been loaded. As far as I am aware, the only two options for XƎTEX are `euler` and `lucbmath`. Unless I've got all confused and the mathtime fonts are not virtual fonts either. But I'm pretty sure they are.

```
\newif\ifzf@math@euler
\newif\ifzf@math@lucida
\newif\ifzf@euler@package@loaded
\@ifpackageloaded{euler}{\zf@euler@package@loadedtrue}
                        {\zf@euler@package@loadedfalse}
\AtBeginDocument{%
  \@ifpackageloaded{euler}{%
    \ifzf@euler@package@loaded
      \zf@math@eulertrue
    \else
      \PackageError{fontspec}{The euler package must be loaded BEFORE fontspec}
        {fontspec only overwrites euler's attempt to\MessageBreak
         define the maths text fonts if fontspec is\MessageBreak
         loaded after euler. Type <return> to proceed\MessageBreak
         with incorrect \protect\mathit, \protect\mathbf, etc}
    \fi}{}
  \@ifpackageloaded{lucbmath}{\zf@math@lucidatrue}{}
  \@ifpackageloaded{lucidabr}{\zf@math@lucidatrue}{}
```

Knuth's CM fonts fonts are all squashed together, combining letters, accents, text symbols and maths symbols all in the one font, `cmr`, plus other things in other fonts. Because we are changing the roman font in the document, we need to redefine all of the maths glyphs in LaTeX's `operators` maths font to still go back to the legacy `cmr` font for all these random glyphs, unless a separate maths font package has been loaded instead.

In every case, the maths accents are always taken from the `operators` font, which is generally the main text font. (Actually, there is a `\hat` accent in `EulerFractur`, but it's *ugly*. So I ignore it. Sorry if this causes inconvenience.)

```
  \DeclareSymbolFont{legacymaths}{OT1}{cmr}{m}{n}
  \SetSymbolFont{legacymaths}{bold}{OT1}{cmr}{bx}{n}
  \DeclareMathAccent{\acute}   {\mathalpha}{legacymaths}{"13}
  \DeclareMathAccent{\grave}   {\mathalpha}{legacymaths}{"12}
  \DeclareMathAccent{\ddot}    {\mathalpha}{legacymaths}{"7F}
  \DeclareMathAccent{\tilde}   {\mathalpha}{legacymaths}{"7E}
  \DeclareMathAccent{\bar}     {\mathalpha}{legacymaths}{"16}
  \DeclareMathAccent{\breve}   {\mathalpha}{legacymaths}{"15}
  \DeclareMathAccent{\check}   {\mathalpha}{legacymaths}{"14}
  \DeclareMathAccent{\hat}     {\mathalpha}{legacymaths}{"5E} % too bad, euler
  \DeclareMathAccent{\dot}     {\mathalpha}{legacymaths}{"5F}
  \DeclareMathAccent{\mathring}{\mathalpha}{legacymaths}{"17}
```

The following symbols are only defined specifically in `euler`, so skip them if that package is loaded.

```
  \ifzf@math@euler\else
    \DeclareMathSymbol{!}{\mathclose}{legacymaths}{"21}
    \DeclareMathSymbol{:}{\mathrel}  {legacymaths}{"3A}
  \DeclareMathSymbol{;}{\mathpunct}{legacymaths}{"3B}
  \DeclareMathSymbol{?}{\mathclose}{legacymaths}{"3F}
```

And these ones are defined both in euler and lucbmath, so we only need to run this code if no extra maths package has been loaded.

lucbmath defines `\colon` as a math symbol, whereas amsmath creates a macro with the same name. We don't want to overwrite amsmath's `\colon`, but we do need to fix up lucbmath's if amsmath isn't loaded.

```
  \ifzf@math@lucida
    \@ifpackageloaded{amsmath}{}{%
      \DeclareMathSymbol{\colon}{\mathpunct}{legacymaths}{"3A}}
  \else
    \DeclareMathSymbol{0}{\mathalpha}{legacymaths}{`0}
    \DeclareMathSymbol{1}{\mathalpha}{legacymaths}{`1}
    \DeclareMathSymbol{2}{\mathalpha}{legacymaths}{`2}
    \DeclareMathSymbol{3}{\mathalpha}{legacymaths}{`3}
    \DeclareMathSymbol{4}{\mathalpha}{legacymaths}{`4}
    \DeclareMathSymbol{5}{\mathalpha}{legacymaths}{`5}
    \DeclareMathSymbol{6}{\mathalpha}{legacymaths}{`6}
    \DeclareMathSymbol{7}{\mathalpha}{legacymaths}{`7}
    \DeclareMathSymbol{8}{\mathalpha}{legacymaths}{`8}
    \DeclareMathSymbol{9}{\mathalpha}{legacymaths}{`9}
    \DeclareMathSymbol{\Gamma}{\mathalpha}{legacymaths}{"00}
    \DeclareMathSymbol{\Delta}{\mathalpha}{legacymaths}{"01}
    \DeclareMathSymbol{\Theta}{\mathalpha}{legacymaths}{"02}
    \DeclareMathSymbol{\Lambda}{\mathalpha}{legacymaths}{"03}
    \DeclareMathSymbol{\Xi}{\mathalpha}{legacymaths}{"04}
    \DeclareMathSymbol{\Pi}{\mathalpha}{legacymaths}{"05}
    \DeclareMathSymbol{\Sigma}{\mathalpha}{legacymaths}{"06}
    \DeclareMathSymbol{\Upsilon}{\mathalpha}{legacymaths}{"07}
    \DeclareMathSymbol{\Phi}{\mathalpha}{legacymaths}{"08}
    \DeclareMathSymbol{\Psi}{\mathalpha}{legacymaths}{"09}
    \DeclareMathSymbol{\Omega}{\mathalpha}{legacymaths}{"0A}
    \DeclareMathSymbol{+}{\mathbin}{legacymaths}{"2B}
    \DeclareMathSymbol{=}{\mathrel}{legacymaths}{"3D}
   \DeclareMathDelimiter{(}{\mathopen} {legacymaths}{"28}{largesymbols}{"00}
   \DeclareMathDelimiter{)}{\mathclose}{legacymaths}{"29}{largesymbols}{"01}
   \DeclareMathDelimiter{[}{\mathopen} {legacymaths}{"5B}{largesymbols}{"02}
   \DeclareMathDelimiter{]}{\mathclose}{legacymaths}{"5D}{largesymbols}{"03}
    \DeclareMathDelimiter{/}{\mathord}{legacymaths}{"2F}{largesymbols}{"0E}
    \DeclareMathSymbol{\mathdollar}{\mathord}{legacymaths}{"24}
  \fi
\fi
```

Finally, we change the font definitions for `\mathrm` and so on. These are defined using the `\zf@rmmaths` (…) macros, which default to `\rmdefault` but may be specified with the `\setmathrm` (…) commands in the preamble.

Since LaTeX only generally defines one level of boldness, we omit `\mathbf` in the bold maths series. It can be specified as per usual with `\setboldmathrm`, which stores the appropriate family name in `\zf@rmboldmaths`.

```
\DeclareSymbolFont{operators}\zf@enc\zf@rmmaths\mddefault\updefault
```

```
1025 \SetSymbolFont{operators}{normal}\zf@enc\zf@rmmaths\mddefault\updefault
1026 \SetMathAlphabet\mathrm{normal}\zf@enc\zf@rmmaths\mddefault\updefault
1027 \SetMathAlphabet\mathit{normal}\zf@enc\zf@rmmaths\mddefault\itdefault
1028 \SetMathAlphabet\mathbf{normal}\zf@enc\zf@rmmaths\bfdefault\updefault
1029 \SetMathAlphabet\mathsf{normal}\zf@enc\zf@sfmaths\mddefault\updefault
1030 \SetMathAlphabet\mathtt{normal}\zf@enc\zf@ttmaths\mddefault\updefault
1031 \SetSymbolFont{operators}{bold}\zf@enc\zf@rmmaths\bfdefault\updefault
1032 \expandafter\ifx\csname zf@rmboldmaths\endcsname\relax
1033   \SetMathAlphabet\mathrm{bold}\zf@enc\zf@rmmaths\bfdefault\updefault
1034   \SetMathAlphabet\mathit{bold}\zf@enc\zf@rmmaths\bfdefault\itdefault
1035 \else
1036   \SetMathAlphabet\mathrm{bold}\zf@enc\zf@rmboldmaths\mddefault\updefault
1037   \SetMathAlphabet\mathbf{bold}\zf@enc\zf@rmboldmaths\bfdefault\updefault
1038   \SetMathAlphabet\mathit{bold}\zf@enc\zf@rmboldmaths\mddefault\itdefault
1039 \fi
1040 \SetMathAlphabet\mathsf{bold}\zf@enc\zf@sfmaths\bfdefault\updefault
1041 \SetMathAlphabet\mathsf{bold}\zf@enc\zf@ttmaths\bfdefault\updefault}
```

The end! Thanks for coming.

1042 ⟨/fontspec⟩

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.