The *fontspec* package

Will Robertson

2008/08/09 v1.18

Contents

1 Introduction

With the introduction of Jonathan Kew's XeTeX,[1] 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`.

Before fontspec, 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 `\emph` or `\bfseries`.

This package provides a completely automatic way to select font families in LaTeX for arbitrary fonts. Furthermore, it allows very flexible control over the selection of advanced font features such as number case and fancy ligatures (and many more!) present in most modern fonts.

1.1 Usage

For basic use, no package options are required:

```
\usepackage{fontspec}% provides font selecting commands
\usepackage{xunicode}% provides unicode character macros
\usepackage{xltxtra} % provides some fixes/extras
```

Ross Moore's xunicode package is highly recommended, as it provides access LaTeX's various methods for accessing extra characters and accents (for example, `\%`, `\$`, `\textbullet`, `\"u`, and so on), plus many more unicode characters.

The xltxtra package adds a couple of general improvements to LaTeX under XeTeX; it also provides the `\XeTeX` macro to typeset the XeTeX logo.

The babel *package is not really supported!* Especially Vietnamese, Greek, and Hebrew at least might not work correctly, as far as I can tell. There's a better chance with Cyrillic and Latin-based languages, however—fontspec ensures at least that fonts should load correctly, but hyphenation and other matters aren't guaranteed.

The rest of this section documents fontspec's package options, which are (briefly):

`cm-default` Don't load the Latin Modern fonts;
`no-math` Don't change any maths fonts;
`no-config` Don't load fontspec.cfg; and,
`quiet` Output fontspec warnings in the log file rather than the console output.

1.1.1 Latin Modern defaults

fontspec defines a new LaTeX font encoding for its purposes to allow the Latin Modern fonts to be used by default. This has three implications:

⋆ v1.12: New!

1. Unicode fonts are loaded by default; it didn't make sense to have the legacy Computer Modern fonts in the Unicode-enabled XeTeX.

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

2. If you don't have the Latin Modern OpenType fonts installed, you might want to consider doing so.

3. fontspec also requires the euenc package[2] to be installed.

Another package option is provided for controlling this behaviour: [cm-default] will ignore the Latin Modern fonts and go about things as it used to. Use this option if you don't have the Latin Modern fonts installed or you (Mac-specifically) want to use the 'default TEX font' without using the xdvipdfmx driver.

1.1.2 Maths 'fiddling'

⋆ v1.14: New!

By default, fontspec adjusts LATEX's default maths setup in order to maintain the correct Computer Modern symbols when the roman font changes. However, it will attempt to avoid doing this if another maths font package is loaded (such as mathpazo or my upcoming unicode-math package).

If you find that it is incorrectly changing the maths font when it should be leaving well enough alone, apply the [no-math] package option to manually suppress its maths font.

1.1.3 Configuration

If you wish to customise any part of the fontspec interface (see later in this manual, Section 7 on page 25 and Section 7.1), this should be done by creating your own fontspec.cfg file,[3] which will be automatically loaded if it is found by XETEX. Either place it in the same folder as the main document for isolated cases, or in a location that XETEX searches by default, *e.g.*, ~/Library/texmf/xelatex/. The package option [no-config] will suppress this behaviour under all circumstances.

⋆ v1.14: Used to be [noconfig], which still works.

1.1.4 Warnings

This package can give many warnings that can be harmless if you know what you're doing. Use the [quiet] package option to write these warnings to the transcript (.log) file instead.

Use the [silent] package option to completely suppress these warnings if you don't even want the .log file cluttered up.

1.2 About this manual

⋆ v1.6: An example warning!

In the unfortunate case that I need to make backwards incompatible changes (you're probably pretty safe these days), such things, and some other comments, are noted in the margin of this document as shown here, with a red star if the change is relevant to the current release of the package. (New features are denoted similarly in blue.)

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

[2] http://tug.ctan.org/cgi-bin/ctanPackageInformation.py?id=euenc

[3] An example is distributed with the package.

documentation if you do not have all of these fonts, many of which are distributed with Mac OS X or are otherwise commercial.

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. Examples whose font type is irrelevant are typeset in green.

2 Brief overview

This manual can get rather in-depth, as there are a lot of font features to cover. A basic preamble set-up is shown below, to simply select some default document fonts. See the file `fontspec-example.tex` for a more detailed example.

```
\usepackage{fontspec}
\defaultfontfeatures{Scale=MatchLowercase}
\setmainfont[Mapping=tex-text]{Baskerville}
\setsansfont[Mapping=tex-text]{Skia}
\setmonofont{Courier}
```

3 Font selection

`\fontspec` `\fontspec[`⟨*font features*⟩`]{`⟨*font name*⟩`}` is the base command of the package, used for selecting the specified ⟨*font name*⟩ in a LaTeX family. The font features argument accepts comma separated ⟨*font feature*⟩=⟨*option*⟩ lists; these will not be fully described until Section 6 on page 14.

As our first example, look how easy it is to select the Hoefler Text typeface with the fontspec package:

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}
  {\scshape\itshape     \pangram}
\bfseries               \pangram
  {\itshape             \pangram}
  {\scshape             \pangram}
  {\itshape\scshape     \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 XeTeX in any way whatsoever: every font that is installed in the operating system may be accessed.

3.1 Default font families

`\setmainfont` `\setsansfont` `\setmonofont` The `\setmainfont`,[4] `\setsansfont`, and `\setmonofont` commands are used to se-

lect 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.

```
\setmainfont{Baskerville}
\setsansfont[Scale=0.86]{Skia}
\setmonofont[Scale=0.8]{Monaco}
\rmfamily\pangram\par
\sffamily\pangram\par
\ttfamily\pangram
```

Here, the scales of the fonts have been chosen to equalise their lowercase letter heights. The Scale font feature will be discussed further in Section 5 on page 10, including methods for automatic scaling.

3.2 Font instances for efficiency

\newfontfamily

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.

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

⋆v1.11: This macro used to be called \newfontinstance. Backwards compatibility is preserved via fontspec.cfg.

This is a *note*.

```
\newfontfamily\notefont{Didot}
\notefont This is a \emph{note}.
```

This macro should be used to create commands that would be used in the same way as \rmfamily, for example.

\newfontface

Sometimes only a specific font face is desired, without accompanying italic or bold variants. This is common when selecting a fancy italic font, say, that has swash features unavailable in the upright forms. \newfontface is used for this purpose:

⋆v1.11: New!

where is all the vegemite

```
\newfontface\fancy
    [Contextuals={WordInitial,WordFinal}]
                        {Hoefler Text Italic}
\fancy where is all the vegemite
```

This example is repeated in Section 6.6 on page 17.

3.3 Arbitrary bold/italic/small caps 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 featuresare 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 features used to be called Bold and Italic, and these shorter names may still be used if you desire.

[4]Or \setromanfont, a historical name that doesn't make much sense when you're, say, typesetting Greek.

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` feature is provided.

⋆ v1.6: `BoldItalic` also works

For those cases that the base font name is repeated, you can replace it with an asterisk (first character only). For example, some space can be saved instead of writing '`Baskerville SemiBold`':

Baskerville *Italic* **SemiBold** ***Italic***

```
\fontspec[BoldFont={* SemiBold}]{Baskerville}
       Baskerville \textit{Italic}
\bfseries SemiBold \textit{Italic}
```

As a matter of fact, this feature can also be used for the upright font too:

Upright ***Italic*** **Bold** ***Bold Italic***

```
\fontspec[UprightFont={* SemiBold},
          BoldFont={* Bold}]{Baskerville}
       Upright \textit{Italic}
\bfseries Bold \textit{Bold Italic}
```

Old-fashioned font families used to distribute their small caps glyphs in separate fonts due to the limitations on the number of glyphs allowed in the PostScript Type 1 format. Such fonts may be used by declaring the `SmallCapsFont` of the family you are specifying:

```
\fontspec[
  SmallCapsFont={Minion MM Small Caps & Oldstyle Figures},
        ]{Minion MM Roman}
Roman 123 \\ \textsc{Small caps 456}
```

3.4 Math(s) fonts

When `\setmainfont`, `\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.[5])

Note that you may find that loading some maths packages won't be as smooth as you expect since fontspec (and XeTeX in general) breaks many of the assumptions of TeX as to where maths characters and accents can be found. Contact me if you have troubles, but I can't guarantee to be able to fix any incompatibilities. The Lucida and Euler maths fonts (the latter loaded with euler rather than eulervm) should be fine; for all others keep an eye out for problems.

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

However, the default text fonts may not necessarily be the ones you wish to

[5] Speaking of euler, if you want to use its `[mathbf]` option, it won't work, and you'll need to put this after fontspec is loaded instead:`\AtBeginDocument{\DeclareMathAlphabet\mathbf{U}{eur}{b}{n}`

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{mathpazo}
\usepackage{fontspec,xunicode}
\setmainfont{Optima}
\setmathrm{Optima}
\setboldmathrm[BoldFont=Optima ExtraBlack]{Optima Bold}
```

and this would allow you to typeset something like this:

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

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

3.5 External fonts

XeTeX v0.995 introduced the feature of loading fonts not installed through the operating system ('external' fonts). This feature is currently only available through the `xdvipdfmx` driver, which is notably *not* the default on Mac OS X.

This feature is handled in `fontspec` with the font feature `ExternalLocation`. When this feature is used, the main argument to `\fontspec` is the *file name* of the font (in contrast to the usual syntax which requires the font display name) and the argument to the feature is the (absolute) path to the font. For example:

```
\fontspec[ExternalLocation=/Users/will/Fonts/]{CODE2000.TTF}
```

If no path is given, then the font will be found in a location normally searched by XeTeX, including the current directory. For example, the following declaration could load either the Latin Modern roman font in the current directory or, say, in `$TEXMF/fonts/opentype/public/lm/`:

```
\fontspec[ExternalLocation]{lmroman10-regular}
```

Bold and italic fonts cannot be automatically selected when external fonts are being used; they must be explicitly declared using the methods described in Section 3.3 on page 5.

3.6 Miscellaneous font selecting details

By the way, from v1.9, `\fontspec` and `\addfontfeatures` will now ignore following spaces as if it were a 'naked' control sequence; *e.g.*, '`M. \fontspec{...} N`' and '`M. \fontspec{...}N`' are the same.

Note that this package redefines the `\itshape` and `\scshape` commands in order to allow them to select italic small caps in conjunction. (This was implicitly shown in the first example, but it's worth mentioning now, too.)

4 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 6 on page 14, but before we look at the range of available font features, it is necessary to discuss how they can be applied.

4.1 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. New calls of `\defaultfontfeatures` overwrite previous ones.

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

```
\fontspec{Didot}
 Some `default' Didot 0123456789            \\
\defaultfontfeatures{Numbers=OldStyle, Colour=888888}
\fontspec{Didot}
 Now grey, with old-style figures: 0123456789
```

4.2 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}
`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`.

4.3 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.

4.4 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.

The font features defined at the top level of the optional `\fontspec` argument are applied to *all* shapes of the family. Using `Upright-`, `SmallCaps-`, `Bold-`, `Italic-`, and `BoldItalicFeatures`, separate font features may be defined to their respective shapes *in addition* to, and with precedence over, the 'global' font features.

Attention All Martini Drinkers
Attention All Martini Drinkers

```
\fontspec{Hoefler Text} \itshape \scshape
Attention All Martini Drinkers \\
\addfontfeature{ItalicFeatures={Alternate = 1}}
Attention All Martini Drinkers \\
```

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

Skia
Skia 'Bold'

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

Note that because most fonts include their small caps glyphs within the main font, these features are applied *in addition* to any other shape-specific features as defined above, and hence `SmallCapsFeatures` can be nested within `ItalicFeatures` and friends. Every combination of upright, italic, bold and small caps can thus be assigned individual features, as shown in the following ludicrous example.

Upright Small Caps
Italic Italic Small Caps
Bold Bold Small Caps
Bold Italic Bold Italic Small Caps

```
\fontspec[
    UprightFeatures={Colour = 220022,
          SmallCapsFeatures = {Colour=115511}},
     ItalicFeatures={Colour = 2244FF,
          SmallCapsFeatures = {Colour=112299}},
       BoldFeatures={Colour = FF4422,
          SmallCapsFeatures = {Colour=992211}},
 BoldItalicFeatures={Colour = 888844,
          SmallCapsFeatures = {Colour=444422}},
        ]{Hoefler Text}
Upright {\scshape Small Caps}\\
\itshape Italic {\scshape Italic Small Caps}\\
\upshape\bfseries Bold {\scshape Bold Small Caps}\\
\itshape Bold Italic {\scshape Bold Italic Small Caps}
```

4.5 Different features for different font sizes

⋆v1.13: New!

The `SizeFeature` featureis a little more complicated than the previous features discussed. It allows different fonts and different font features to be selected for a given font family as the point size varies.

Input	Font size, s
`Size = X-`	$s \geq$ `X`
`Size = -Y`	$s <$ `Y`
`Size = X-Y`	`X` $\leq s <$ `Y`
`Size = X`	$s =$ `X`

Table 1: Syntax for specifying the size to apply custom font features.

It takes a comma separated list of braced, comma separated lists of features for each size range. Each sub-list must contain the `Size` option to declare the size range, and optionally `Font` to change the font based on size. Other (regular) fontspec features that are added are used on top of the font features that would be used anyway.


```
\fontspec[ SizeFeatures={
    {Size={-8}, Font=Apple Chancery, Colour=AA0000},
    {Size={8-14}, Colour=00AA00},
    {Size={14-}, Colour=0000AA}} ]{Skia}
{\scriptsize Small\par} Normal size\par {\Large Large\par}
```

A less trivial example is shown in the context of optical font sizes in Section 6.2 on page 14.

To be precise, the `Size` sub-feature accepts arguments in the form shown in Table 1. Braces around the size range are optional. For an exact font size (`Size=X`) font sizes chosen near that size will 'snap'. For example, for size definitions at exactly 11pt and 14pt, if a 12pt font is requested *actually* the 11pt font will be selected. This is a remnant of the past when fonts were designed in metal (at obviously rigid sizes) and later when bitmap fonts were similarly designed for fixed sizes.

If additional features are only required for a single size, the other sizes must still be specified. As in:

```
SizeFeatures={
   {Size=-10,Numbers=Uppercase},
   {Size=10-}}
```

Otherwise, the font sizes greater than 10 won't be defined!

5 Font independent options

Features introduced in this section may be used with any font.

5.1 Scale

In its explicit form, `Scale` takes a single numeric argument for linearly scaling the font, as demonstrated in Section 3.1 on page 4. Since version 0.99 of XeTeX, however, it is now possible to measure the correct dimensions of the fonts loaded, and hence calculate values to scale them automatically.

⋆ v1.9: As of Dec. 2005

The `Scale` feature nowalso takes the options `MatchLowercase` and `MatchUp-`

percase, which will scale the font being selected to match the current default roman font to either the height of the lowercase or uppercase letters, respectively.

The perfect match is hard to find.
L O G O F O N T

```
\setmainfont{Georgia}
\newfontfamily\lc[Scale=MatchLowercase]{Verdana}
 The perfect match {\lc is hard to find.}\\
\newfontfamily\uc[Scale=MatchUppercase]{Arial}
 L O G O \uc F O N T
```

The amount of scaling used in each instance is reported in the .log file. Since there is some subjectivity about the exact scaling to be used, these values should be used to fine-tune the results.

5.2 Mapping

Mapping enables a XeTeX text-mapping scheme.

“¡A small amount of—text!”

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

5.3 Colour

Colour (or Color), also shown in Section 4.1 on page 8 and Section 6 on page 14, uses XeTeX font specifications to set the colour of the text. 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.)

```
\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}
```

5.4 Interword space

While the space between words can be varied on an individual basis with the TeX primitive \spaceskip command, it is more convenient to specify this information when the font is first defined.

The space in between words in a paragraph will be chosen automatically by XeTeX, and generally will not need to be adjusted. For those times when the precise details are important, the WordSpace features is provided, which takes either a single scaling factor to scale the value that XeTeX has already chosen, or a triplet of comma-separated values for the nominal value, the stretch, and the shrink of the interword space, respectively. *I.e.*, WordSpace=0.8 is the same as WordSpace={0.8,0.8,0.8}.

For example, I believe that the Cochin font, as distributed with Mac OS X, is too widely spaced. Now, this can be rectified, as shown below.

Some filler text for our example to take up some space, and to demonstrate the large default interword space in *Cochin*.

```
\fontspec{Cochin}
\fillertext
\vspace{1em}
```

Some filler text for our example to take up some space, and to demonstrate the large default interword space in *Cochin*.

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

Be careful with the unpredictable things that the AAT font renderer can do with the text! Unlike TEX, Mac OS X will allow fonts to letterspace themselves, which can be seen above; OpenType fonts, however, will not show this tendency, as they do not support this arguably dubious feature.

5.5 Post-punctuation space

If `\frenchspacing` is *not* in effect, TEX will allow extra space after some punctuation in its goal of justifying the lines of text. Generally, this is considered old-fashioned, but occasionally in small amounts the effect can be justified, pardon the pun.

The `PunctuationSpace` feature takes a scaling factor by which to adjust the nominal value chosen for the font. Note that `PunctuationSpace=0` is *not* equivalent to `\frenchspacing`, although the difference will only be apparent when a line of text is under-full.

Letters, Words. Sentences.
Letters, Words. Sentences.
Letters, Words. Sentences.

```
\nonfrenchspacing
\fontspec{Baskerville}
 Letters, Words. Sentences.          \par
\fontspec[PunctuationSpace=0.5]{Baskerville}
 Letters, Words. Sentences.          \par
\fontspec[PunctuationSpace=0]{Baskerville}
 Letters, Words. Sentences.
```

Also be aware that the above caveat for interword space also applies here, so after the last line in the above example, the `PunctuationSpace` for *all* Baskerville instances will be `0`.

5.6 Letter spacing

Letter spacing, or tracking, is the term given to adding (or subtracting) a small amount of horizontal space in between adjacent characters. It is specified with the `LetterSpace`, which takes a numeric argument.

The letter spacing parameter is a normalised additive factor (not a scaling factor); it is defined as a percentage of the font size. That is, for a 10 pt font, a letter spacing parameter of '`1.0`' will add 0.1 pt between each letter.

USE TRACKING FOR DISPLAY CAPS TEXT
USE TRACKING FOR DISPLAY CAPS TEXT

```
\fontspec{Didot}
\addfontfeature{LetterSpace=0.0}
USE TRACKING FOR DISPLAY CAPS TEXT \\
\addfontfeature{LetterSpace=2.0}
USE TRACKING FOR DISPLAY CAPS TEXT
```

This functionality *should not be used for lowercase text*, which is spacing correctly to begin with, but it can be very useful, in small amounts, when setting small caps or all caps titles. Also see the OpenType `Uppercase` option of the `Letters` feature (Section 6.4 on page 16).

5.7 The hyphenation character

The letter used for hyphenation may be chosen with the `HyphenChar` feature. It takes three types of input, which are chosen according to some simple rules. If the input is the string `None`, then hyphenation is suppressed for this font. If the input is a single character, then this character is used. Finally, if the input is longer than a single character it must be the UTF-8 slot number of the hyphen character you desire.

Below, Adobe Garamond Pro's uppercase hyphenation character[6] is used to demonstrate a possible use for this feature. The second example redundantly demonstrates the default behaviour of using the hyphen as the hyphenation character.

A MULTITUDE OF OBSTREPEROUSLY HYPHENATED ENTITIES

A MULTITUDE OF OBSTREPER-OUSLY HYPHENATED ENTITIES

A MULTITUDE OF OBSTREPER-OUSLY HYPHENATED ENTITIES

```
\def\text
  {A MULTITUDE OF OBSTREPEROUSLY HYPHENATED ENTITIES
                                        \par\vspace{1ex}}
\fontspec[HyphenChar=None]{Adobe Garamond Pro} \text
\fontspec[HyphenChar={-}]{Adobe Garamond Pro} \text
\fontspec[HyphenChar="F6BA]{Adobe Garamond Pro} \text
```

Note that in an actual situation, the `Uppercase` option of the `Letters` feature would probably supply this for you (see Section 6.4 on page 16).

The `xltxtra` package redefines LaTeX's `\-` macro such that it adjusts along with the above changes.

5.8 Font transformations

In rare situations users may want to mechanically distort the shapes of the glyphs in the current font. Please don't overuse these features; they can be extremely ugly if overused.

ABCxyz *ABCxyz*

```
\fontspec{Charis SIL} \emph{ABCxyz} \quad
\fontspec[FakeSlant=0.2]{Charis SIL} ABCxyz
```

ABCxyz ABCxyz

```
\fontspec{Charis SIL}  ABCxyz \quad
\fontspec[FakeStretch=1.2]{Charis SIL} ABCxyz
```

ABCxyz **ABCxyz**

```
\fontspec{Charis SIL} \textbf{ABCxyz} \quad
\fontspec[FakeBold=1.5]{Charis SIL} ABCxyz
```

If values are omitted, their defaults are as shown above.

[6] I found the character, and its number, in Mac OS X's Character Palette.

6 Font-dependent 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:


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

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 XeTeX will simply use the last option that is specified (in this case using `Lining` over `OldStyle`).

If a feature or an option is requested that the font does not have, a warning is given in the console output. As mentioned in 1.1.4 on page 3 these warnings can be suppressed by selecting the `[quiet]` package option.

6.1 Different font technologies: AAT and ICU

XeTeX supports two rendering technologies for typesetting, selected with the `Renderer` font feature. The first, `AAT`, is that provided (only) by Mac OS X itself. The second, `ICU`, is an open source OpenType interpreter. It provides much greater support for OpenType features, notably contextual arrangement, over `AAT`.

In general, this feature will not need to be explicitly called: for OpenType fonts, the `ICU` renderer is used automatically, and for AAT fonts, `AAT` is chosen by default. Some fonts, however, will contain font tables for *both* rendering technologies, such as the Hiragino Japanese fonts distributed with Mac OS X, and in these cases the choice may be required.

Among some other font features only available through a specific renderer, `ICU` provides for the `Script` and `Language` features, which allow different font behaviour for different alphabets and languages; see Section 6.19 on page 23 for the description of these features. *Because these font features can change which features are able to be selected for the font instance, they are selected by* `fontspec` *before all others and will automatically and without warning select the* `ICU` *renderer.*

6.2 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.

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, no optical size font substitution is performed:

```
\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
```

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

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

Automatic optical size
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 6.18 on page 23 for further 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 continuous 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.

```
\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                     \\
```

The `SizeFeatures` feature (Section 4.5 on page 9) can be used to specify exactly which optical sizes will be used for ranges of font size. For example, something like

```
\fontspec[
  SizeFeatures={
    {Size=-10,     OpticalSize=8 },
    {Size= 10-14,  OpticalSize=10},
    {Size= 14-18,  OpticalSize=14},
    {Size=    18-, OpticalSize=18}}
       ]{Warnock Pro}
```

6.3 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]{Adobe Garamond Pro}
 \textit{strict firefly}                \\
\fontspec[Ligatures=NoCommon]{Adobe Garamond Pro}
 \textit{strict firefly}
```

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 XETEX; you must use a 0 pt kern or `\hbox` (*e.g.*, `\null`) to split the characters up.

6.4 Letters

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

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

⋆ v1.9: The `Uppercase...` variants have changed (*e.g.*, from `SMALLCAPS`) to allow for more flexible option handling in the future. The old forms still work, for now…

OpenType fonts have some different options: `Uppercase`, `SmallCaps`, `PetiteCaps`, `UppercaseSmallCaps`, `UppercasePetiteCaps`, 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'). 'Unicase' is a weird hybrid of upper and lower case letters.

THIS SENTENCE NO VERB
THIS SENTENCE NO verb

```
\fontspec[Letters=SmallCaps]{TeX Gyre Adventor}
 THIS SENTENCE no verb                \\
\fontspec[Letters=UppercaseSmallCaps]{TeX Gyre Adventor}
 THIS SENTENCE no verb
```

The `Uppercase` option is also provided *but* it will (probably) not actually map letters to uppercase.[7] It will, however, select various uppercase forms for glyphs such as accents and dashes.

UPPER-CASE EXAMPLE
UPPER-CASE EXAMPLE

```
\fontspec{Warnock Pro}
 UPPER-CASE EXAMPLE \\
\addfontfeature{Letters=Uppercase}
 UPPER-CASE EXAMPLE
```

The `Kerning` feature also contains an `Uppercase` option, which adds a small amount of spacing in between letters (see Section 6.13 on page 21). This feature was originally planned to be included with the one above (so `Letters=Uppercase` would do both punctuation *and* tracking), but I decided that it would be a bad idea to break the one-to-one correspondence with `fontspec` and OpenType features. (Sorry TUGboat readers!)

[7] If you want automatic uppercase letters, look to LATEX's `\MakeUppercase` command.

6.5 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 4.2 on page 8.

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 0̸123456789

```
\fontspec[Numbers=Lining]{TeX Gyre Bonum}
 0123456789
\fontspec[Numbers=SlashedZero]{TeX Gyre Bonum}
 0123456789
```

6.6 Contextuals

This feature refers to glyph substitution that vary by their position; things like contextual swashes are implemented here. The options for AAT fonts are `WordInitial`, `WordFinal`, `LineInitial`, `LineFinal`, and `Inner` (also called 'non-final' sometimes). As non-exclusive selectors, like the ligatures, you can turn them off by prefixing their name with `No`.

⋆ v1.9: This feature used to be called `Swashes`. This name still works, for now.

where is all the vegemite

```
\newfontface\fancy
    [Contextuals={WordInitial,WordFinal}]
                        {Hoefler Text Italic}
\fancy where is all the vegemite
```

'Inner' ſwaſhes can *ſometimes* contain the archaic long s.

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

For OpenType fonts, all feature options as above but the `LineInitial` feature are supported, and `Swash` turns on contextual swashes.

⋆ v1.9: Used to be `Contextual`; still works.

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

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

Historic forms (*e.g.*, long s as shown above) are accessed in OpenType fonts via the feature `Style=Historic`; this is generally *not* contextual in OpenType, which is why it is not included here.

⋆ v1.18: New!

New option `Alternate` corresponds to the raw feature `calt`. It should usually be activated by default.

6.7 Vertical position

The `VerticalPosition` feature is used to access things like subscript (`Inferior`) and superscript (`Superior`) numbers and letters (and a small amount of punctuation, sometimes). The `Ordinal` option is (supposed to be) contextually sensitive to only raise characters that appear directly after a number.

Normal superior inferior
1st 2nd 3rd 4th 0th 8abcde

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

```
\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
```

The `xltxtra` package redefines the `\textsubscript` and `\textsuperscript` commands to use the above font features.

6.8 Fractions

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.

⋆v1.7: This feature has changed: no backwards compatibility!

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

½ 5/6
1/2 5/6

```
\fontspec[Fractions=On]{Skia}
 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:

13579/24680
13579/24680

```
\fontspec[Fractions=Diagonal]{Skia}
      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) also provide for the `Alternate` feature:

1/2 1/4 5/6 13579/24680
1/2 1/4 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 \\
```

The xltxtra package provides a `\vfrac` command for creating arbitrary so-called 'vulgar' fractions:

13579⁄24680

```
\fontspec{Warnock Pro}
\vfrac{13579}{24680}
```

6.9 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.

```
\newcounter{var}\newcounter{trans}
\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', again specified numerically. I don't have a font to demonstrate this feature with, unfortunately. See Section 7 on page 25 for a way to assign names to variants, which should be done on a per-font basis.

6.10 Alternates

Selection of `Alternates` *again* must be done numerically. Here's an example with an AAT font:

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}
```

See Section 7 on page 25 for a way to assign names to alternates, which should be done on a per-font basis.

For OpenType fonts, this option is used to access numerical variations of the raw `salt` feature. I can't show an example, but here's how it would be used:

⋆v1.18: New!

```
\fontspec[Alternate=1]{Garamond Premier Pro}
```

Numbering starts from `0` for the first stylistic alternate. Note that the `Style=Alternate` option is equivalent to `Alternate=0` to access the default case.

6.11 Style

The options of the Style feature are defined in AAT as one of the following: Display, Engraved, IlluminatedCaps, Italic, Ruby,[8] TallCaps, or TitlingCaps.

ICU supported options are Alternate, Italic, Historic, Ruby,[8] Swash, TitlingCaps, HorizontalKana, and VerticalKana.

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
```

Note the occasional inconsistency with which font features are labelled; a long-tailed 'Q' could turn up anywhere!

M Q Z
M Q Z

```
\fontspec{Adobe Jenson Pro}
 M Q Z                            \\
\addfontfeature{Style=Historic}
 M Q Z
```

TITLING CAPS
TITLING CAPS

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

Two features in one example; Italic affects the Latin text and Ruby the Japanese:

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
```

Note the difference here between the default and the horizontal style kana:

ようこそ ワカヨタレソ
ようこそ ワカヨタレソ
ようこそ ワカヨタレソ

```
\fontspec{Hiragino Mincho Pro}
  ようこそ ワカヨタレソ  \\
{\addfontfeature{Style=HorizontalKana}
  ようこそ ワカヨタレソ} \\
{\addfontfeature{Style=VerticalKana}
  ようこそ ワカヨタレソ}
```

6.12 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.

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

No options for OpenType fonts.

[8] 'Ruby' refers to a small optical size, used in Japanese typography for annotations.

6.13 Kerning

Well designed fonts contain kerning information that controls the spacing between letter pairs, on an individual basis. The `Kerning` feature provides options to control this, for OpenType fonts only.

The options provided for now are `On`, `Off` (don't know why you'd want to), and `Uppercase`.

Ta AV
Ta AV

```
\fontspec{Warnock Pro}
 Ta AV                \\
\fontspec[Kerning=Off]{Warnock Pro}
 Ta AV
```

As briefly mentioned previously at the end of Section 6.4 on page 16, the `Uppercase` option will add a small amount of tracking between uppercase letters:

UPPER-CASE EXAMPLE
UPPER-CASE EXAMPLE

```
\fontspec{Warnock Pro}
 UPPER-CASE EXAMPLE \\
\addfontfeature{Kerning=Uppercase}
 UPPER-CASE EXAMPLE
```

6.14 CJK shape

There have been many standards for how CJK ideographic glyphs are 'supposed' to look. Some fonts will contain 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 `CJKShape` options: `Traditional`, `Simplified`, `JIS1978`, `JIS1983`, `JIS1990`, and `Expert`. OpenType also supports the `NLC` option.

⋆ v1.9: Was `CharacterShape`, which wasn't very descriptive. **No** backwards compatibility.

唖噛躯 妍并訝
唖噛躯 妍幷訝
啞嚙軀 妍并訝

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

6.15 Character width

Many Asian fonts are equipped with variously spaced characters for shoehorning into their generally monospaced text. These are accessed through the `CharacterWidth` feature.[9] For now, OpenType and AAT share the same six options for this feature: `Proportional`, `Full`, `Half`, `Third`, `Quarter`, `AlternateProportional`, and `AlternateHalf`. AAT also allows `Default` to return to whatever was originally specified.

⋆ v1.9: Was `TextSpacing`, which wasn't very descriptive. **No** backwards compatibility.

[9] 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.

Japanese alphabetic glyphs (in 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
ようこそ　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}
{\addfontfeature{CharacterWidth=Proportional}\test}\\
{\addfontfeature{CharacterWidth=Full}\test}\\
{\addfontfeature{CharacterWidth=Half}\test}
```

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

—１２３２１—
-1234554321-
-123456787654321-
-1234567890987654321-

```
\fontspec[Renderer=AAT]{Hiragino Mincho Pro}
{\addfontfeature{CharacterWidth=Full}
 ---12321---}\\
{\addfontfeature{CharacterWidth=Half}
 ---1234554321---}\\
{\addfontfeature{CharacterWidth=Third}
 ---123456787654321---}\\
{\addfontfeature{CharacterWidth=Quarter}
 ---12345678900987654321---}
```

The option `CharacterWidth=Full` doesn't work with the default OpenType font renderer (ICU) due to a bug in the Hiragino fonts.

6.16 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
```

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}
 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 XeTeX can access alternate annotation forms, even if they exist (as in this case) in the font.

6.17 Vertical typesetting

XeTeX provides for vertical typesetting simply with the ability to rotate the individual glyphs as a font is used for typesetting.

共産主義者は

共産主義者は

```
\fontspec{Hiragino Mincho Pro}
共産主義者は

\fontspec[Renderer=AAT,Vertical=RotatedGlyphs]{Hiragino Mincho Pro}
\rotatebox{-90}{共産主義者は}% requires the graphicx package
```

No actual provision is made for typesetting top-to-bottom languages; for an example of how to do this, see the vertical Chinese example provided in the XeTeX documentation.

6.18 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 can have any bold weight you like between the two extremes.

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

Really light and extended Skia
Really fat and condensed Skia

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

Variations along a multiple master font's optical size axis has been shown previously in Section 6.2 on page 14.

6.19 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 2 on page 25 and Table 3 on page 26, respectively. When a script or language is requested that is not supported by the current font, a warning is printed in the console output.

Because these font features can change which features are able to be selected for the font, they are selected by `fontspec` before all others and will specifically select the `ICU` renderer for this font, as described in Section 6.1 on page 14.

6.19.1 `Script` examples

In the following examples, the same font is used to typeset the verbatim input and the XeTeX 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}
العربي
```

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
```

הִתְרַחֲקוּ

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

6.19.2 `Language` examples

Vietnamese requires careful diacritic placement:

cấp số mỗi
cấp số mỗi

```
\fontspec{Doulos SIL}
 cấp số mỗi \\
\addfontfeature{Language=Vietnamese}
 cấp số mỗi
```

Moldavian, as a typical example from Ralf Stubner's FPL Neu font:

Ş ş Ţ ţ
Ş ş Ţ ţ

```
\fontspec{FPL Neu}
 Ş ş Ţ ţ  \\
\addfontfeature{Language=Moldavian}
 Ş ş Ţ ţ
```

6.19.3 Defining new scripts and languages

\newfontscript
\newfontlanguage

Further scripts and languages may be added with the `\newfontscript` and `\newfontlanguage` commands. For example,

```
\newfontscript{Arabic}{arab}
\newfontlanguage{Turkish}{TUR}
```

The first argument is the fontspec name, the second the OpenType definition. The advantage to using these commands rather than `\newfontfeature` (see Section 7) is the error-checking that is performed when the script or language is requested.

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 2: Defined `Scripts` for OpenType fonts. Aliased names are shown in adjacent positions marked with red pilcrows (¶), defined in `fontspec.cfg`.

7 Defining new features

This package cannot hope to contain every possible font feature. Three 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.

\newAATfeature

New AAT features may be created with this command:

`\newAATfeature{`⟨*feature*⟩`}{`⟨*option*⟩`}{`⟨*feature code*⟩`}{`⟨*selector code*⟩`}`

Use the XeTeX file `AAT-info.tex` to obtain the code numbers. For example:

```
\newAATfeature{Alternate}{HoeflerSwash}{17}{1}
\fontspec[Alternate=HoeflerSwash]{Hoefler Text Italic}
 This is XeTeX by Jonathan Kew.
```

This is XeTeX by Jonathan Kew.

This command replaces `\newfeaturecode`, which is provided for backwards compatibility via `fontspec.cfg`.

\newICUfeature

New OpenType features may be created with this command:

`\newICUfeature{`⟨*feature*⟩`}{`⟨*option*⟩`}{`⟨*feature tag*⟩`}`

In the following example, the Moldavian language (see Section 6.19 on page 23) must be activated to achieve the effect shown.

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

Table 3: Defined Languages for OpenType fonts. Note that they are sorted alphabetically *not* by name but by OpenType tag, which is a little irritating, really.

Ş ş Ţ ţ
Ş ş Ţ ţ

```
\newICUfeature{Style}{NoLocalForms}{-locl}
\fontspec[Language=Moldavian]{FPL Neu}
Ş ş Ţ ţ  \\
\addfontfeature{Style=NoLocalForms}
Ş ş Ţ ţ
```

\newfontfeature

In case the above commands do not accommodate the desired font feature (perhaps a new XeTeX feature that fontspec hasn't been updated to support), a command is provided to pass arbitrary input into the font selection string:

\newfontfeature{⟨*name*⟩}{⟨*input string*⟩}

For example, Zapfino contains the feature 'Avoid d-collisions'. To access it with this package, you could do the following:

```
\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
```

The advantage to using the \newAATfeature and \newICUfeature commands is that they check if the selected font actually contains the font feature. By contrast, \newfontfeature will not give a warning for improper input.

7.1 Renaming existing features & options

\aliasfontfeature

If you don't like the name of a particular font feature, it may be aliased to another with the \aliasfontfeature{⟨*existing name*⟩}{⟨*new name*⟩} command:

Roman Letters *And Swash*

```
\aliasfontfeature{ItalicFeatures}{IF}
\fontspec[IF = {Alternate=1}]{Hoefler Text}
Roman Letters \itshape And Swash
```

Spaces in feature (and option names, see below) *are* allowed. (You may have noticed this already in the lists of OpenType scripts and languages).

\aliasfontfeatureoption

If you wish to change the name of a font feature option, it can be aliased to another with the command \aliasfontfeatureoption{⟨*font feature*⟩}{⟨*existing name*⟩}{⟨*new name*⟩}:

Scientific Inferior: $_{12345}$

```
\aliasfontfeature{VerticalPosition}{Vert Pos}
\aliasfontfeatureoption{VerticalPosition}{ScientificInferior}{Sci Inf}
\fontspec[Vert Pos=Sci Inf]{Warnock Pro}
 Scientific Inferior: 12345
```

This example demonstrates an important point: when aliasing the feature options, the *original* feature name must be used when declaring to which feature the option belongs.

Only feature options that exist as sets of fixed strings may be altered in this way. That is, Proportional can be aliased to Prop in the Letters feature, but 550099BB cannot be substituted for Purple in a Colour specification. For this type of thing, the \newfontfeature command should be used to declare a new, *e.g.*, PurpleColour feature:

```
\newfontfeature{PurpleColour}{color=550099BB}
```

7.2 Going behind fontspec's back

⋆v1.13: New!

Expert users may wish not to use fontspec's feature handling at all, while still taking advantage of its LaTeX font selection conveniences. The RawFeaturefont feature allows literal XeTeX font feature selection when you happen to have the OpenType feature tag memorised.

FPL Neu small caps

```
\fontspec[RawFeature=+smcp]{FPL Neu}
FPL Neu small caps
```

Multiple features can either be included in a single declaration:

```
[RawFeature=+smcp;+onum]
```

or with multiple declarations:

```
[RawFeature=+smcp, RawFeature=+onum]
```

File I
fontspec.sty

8 Implementation

Herein lie 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 them together at the time. (Possibly inspired by Hermann *Zapf*.)

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

```
\RequirePackage{ifxetex}
\RequireXeTeX
```

8.1 Bits and pieces

Conditionals

```
\newif\ifzf@firsttime
\newif\ifzf@nobf
\newif\ifzf@noit
\newif\ifzf@nosc
\newif\ifzf@tfm
\newif\ifzf@atsui
\newif\ifzf@icu
\newif\ifzf@mm
```

For dealing with legacy maths

```
\newif\ifzf@math@euler
\newif\ifzf@math@lucida
\newif\ifzf@package@euler@loaded
```

For package options:

```
\newif\if@zf@configfile
\newif\if@zf@euenc
\newif\if@zf@math
```

Counters

```
\newcount\c@zf@newff
\newcount\c@zf@index
\newcount\c@zf@script
\newcount\c@zf@language
```

fontspec shorthands:

```
\def\zf@nl{^^J\space\space\space\space}
\newcommand\zf@PackageError[2]{%
  \PackageError{fontspec}{{\zf@nl #1}^^J}{#2}}
\newcommand\zf@PackageWarning[1]{%
  \PackageWarning{fontspec}{\zf@nl #1^^JThis warning occured}}
\newcommand\zf@PackageInfo[1]{\PackageInfo{fontspec}{#1}}
```

\def@cx
\gdef@cx
\let@cc

LaTeX3–like syntax for various low level commands. Makes life much easier; can't wait for the official interface :)

```
\providecommand\def@cx[2]{\expandafter\edef\csname#1\endcsname{#2}}
\providecommand\gdef@cx[2]{\expandafter\xdef\csname#1\endcsname{#2}}
\providecommand\let@cc[2]{%
  \expandafter\let\csname#1\expandafter\endcsname\csname#2\endcsname}
```

8.2 Option processing

```
\DeclareOption{cm-default}{\@zf@euencfalse}
\DeclareOption{lm-default}{\@zf@euenctrue}
\DeclareOption{math}{\@zf@mathtrue}
\DeclareOption{no-math}{\@zf@mathfalse}
\DeclareOption{config}{\@zf@configfiletrue}
\DeclareOption{no-config}{\@zf@configfilefalse}
\DeclareOption{noconfig}{\@zf@configfilefalse}
\DeclareOption{quiet}{%
  \let\zf@PackageWarning\zf@PackageInfo
  \let\zf@PackageInfo\@gobble}
\DeclareOption{silent}{%
  \let\zf@PackageWarning\@gobble
  \let\zf@PackageInfo\@gobble}
\ExecuteOptions{config,lm-default,math}
\ProcessOptions*
```

Only proceed if it is XeTeX that is doing the typesetting:

```
\RequirePackage{ifxetex}
\RequireXeTeX
```

8.3 Packages

We require the calc package for autoscaling and a recent version of the xkeyval package for option processing.

```
\RequirePackage{calc}
\RequirePackage{xkeyval}[2005/05/07]
```

8.4 Encodings

Frank Mittelbach has recommended using the 'EUx' family of font encodings to experiment with unicode. Now that XeTeX can find fonts in the texmf tree, the Latin Modern OpenType fonts can be used as the defaults. See the euenc collection of files for how this is implemented.

```
\if@zf@euenc
  \def\zf@enc{EU1}
  \renewcommand{\rmdefault}{lmr}
  \renewcommand{\sfdefault}{lmss}
  \renewcommand{\ttdefault}{lmtt}
  \RequirePackage[\zf@enc]{fontenc}
\else
  \def\zf@enc{U}
  \let\encodingdefault\zf@enc
```

```
\fi
\let\UTFencname\zf@enc
```

Dealing with a couple of the problems introduced by babel:

```
\let\cyrillicencoding\zf@enc
\let\latinencoding\zf@enc
\g@addto@macro\document{%
  \let\cyrillicencoding\zf@enc
  \let\latinencoding\zf@enc}
```

That latin encoding definition is repeated to suppress font warnings. Something to do with `\select@language` ending up in the `.aux` file which is read at the beginning of the document.

8.5 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 8.6 on page 35.

8.5.1 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
  \ignorespaces}
```

`\setmainfont` `\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*\setmainfont[2][]{%
  \zf@fontspec{#1}{#2}%
  \let\rmdefault\zf@family
  \normalfont}
\let\setromanfont\setmainfont
\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}
```

\newfontfamily
\newfontface

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. This family is then used in a typical NFSS \fontfamily declaration, saved in the macro name specified.

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

\newfontface uses an undocumented feature of the BoldFont feature; if its argument is empty (*i.e.*, BoldFont={}), then no bold font is searched for.

```
\newcommand*\newfontface[1]{%
  \@ifnextchar[{\newfontface@i#1}{\newfontface@i#1[]}}
\def\newfontface@i#1[#2]#3{%
  \zf@fontspec{BoldFont={},ItalicFont={},SmallCapsFont={},#2}{#3}%
  \edef\@tempa{%
    \noexpand\DeclareRobustCommand\noexpand#1
```

```
      {\noexpand\fontfamily{\zf@family}\noexpand\selectfont}}%
  \@tempa}
```

8.5.2 Font feature selection

`\defaultfontfeatures` 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 created, 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 `\zf@default@options` is emptied inside the group; this is allowed as `\zf@family` is globally defined in `\zf@fontspec`), 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]{%
  \ifcsname zf@family@fontdef\f@family\endcsname
    \begingroup
      \let\zf@default@options\@empty
      \edef\@tempa{%
        \noexpand\zf@fontspec
          {\csname zf@family@options\f@family\endcsname,#1}%
          {\csname zf@family@fontname\f@family\endcsname}}%
      \@tempa
    \endgroup
    \fontfamily\zf@family\selectfont
  \else
    \zf@PackageWarning{%
      \protect\addfontfeature (s) ignored;\zf@nl
      it cannot be used with a font that wasn't selected by fontspec.}%
  \fi
  \ignorespaces}
\let\addfontfeature\addfontfeatures
```

8.5.3 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.

```
\newcommand*\newfontfeature[2]{%
  \stepcounter{zf@newff}%
  \def@cx{+zf-#1}{+zf-\the\c@zf@newff}%
  \define@key[zf]{options}{#1}[]{%
    \zf@update@family{\csname+zf-#1\endcsname}%
    \zf@update@ff{#2}}}
```

\newAATfeature This command assigns a new AAT feature by its code (#2,#3) to a new name (#1). Better than \newfontfeature because it checks if the feature exists in the font it's being used for.

```
\newcommand*\newAATfeature[4]{%
  \unless\ifcsname zf@options@#1\endcsname
    \zf@define@font@feature{#1}%
  \fi
  \key@ifundefined[zf]{#1}{#2}{}{%
    \zf@PackageWarning{Option '#2' of font feature '#1' overwritten.}}%
  \zf@define@feature@option{#1}{#2}{#3}{#4}{}}
```

\newICUfeature This command assigns a new OpenType feature by its abbreviation (#2) to a new name (#1). Better than \newfontfeature because it checks if the feature exists in the font it's being used for.

```
\newcommand*\newICUfeature[3]{%
  \unless\ifcsname zf@options@#1\endcsname
    \zf@define@font@feature{#1}%
  \fi
  \key@ifundefined[zf]{#1}{#2}{}{%
    \zf@PackageWarning{Option '#2' of font feature '#1' overwritten.}}%
  \zf@define@feature@option{#1}{#2}{}{}{#3}}
```

\aliasfontfeature
\aliasfontfeatureoption User commands for renaming font features and font feature options. Provided I've been consistent, they should work for everything.

```
\newcommand*\aliasfontfeature[2]{\multi@alias@key{#1}{#2}}
\newcommand*\aliasfontfeatureoption[3]{%
  \keyval@alias@key[zf@feat]{#1}{#2}{#3}}
```

\newfontscript Mostly used internally, but also possibly useful for users, to define new OpenType 'scripts', mapping logical names to OpenType script tags. Iterates though the scripts in the selected font to check that it's a valid feature choice, and then prepends the (XeTeX) \font feature string with the appropriate script selection tag.

```
\newcommand*\newfontscript[2]{%
  \define@key[zf@feat]{Script}{#1}[]{%
    \zf@check@ot@script{#2}%
    \if@tempswa
      \global\c@zf@script\@tempcnta\relax
      \xdef\zf@script@name{#1}%
```

```
      \xdef\zf@family@long{\zf@family@long+script=#1}%
      \xdef\zf@pre@ff{script=#2,\zf@pre@ff}%
    \else
      \zf@PackageWarning{Font \fontname\zf@basefont\space does not contain script '#1'}%
    \fi}}
```

`\newfontlanguage` Mostly used internally, but also possibly useful for users, to define new OpenType 'languages', mapping logical names to OpenType language tags. Iterates though the languages in the selected font to check that it's a valid feature choice, and then prepends the (XeTeX) `\font` feature string with the appropriate language selection tag.

```
\newcommand*\newfontlanguage[2]{%
  \define@key[zf@feat]{Lang}{#1}[]{%
    \zf@check@ot@lang{#2}%
    \if@tempswa
      \global\c@zf@language\@tempcnta\relax
      \xdef\zf@language@name{#1}%
      \xdef\zf@family@long{\zf@family@long+lang=#1}%
      \xdef\zf@pre@ff{\zf@pre@ff language=#2,}%
    \else
      \zf@PackageWarning{%
        Font \fontname\zf@basefont\space does not contain
        language '#1' for script '\zf@script@name'}%
    \fi}}
```

8.6 Internal macros

`\zf@fontspec` This is the command that defines font families for use, the underlying procedure of all `\fontspec`-like commands. Given a list of font features (`#1`) for a requested font (`#2`, stored in `\zf@fontname` globally for the `\zf@make@aat@feature@string` macro), it will define an NFSS family for that font and put the family name into `\zf@family`.

This macro does its processing inside a group, but it's a bit worthless coz there's all sorts of `\global` action going on. Pity. Anyway, lots of things are branched out for the pure reason of splitting the code up into logical chunks. Some of it is never even re-used, so it all might be a bit obfuscating. (*E.g.*, `\zf@init` and `\zf@set@font@type`.)

First off, initialise some bits and pieces and run the `preparse` feature processing. This catches font features such as `Renderer` that can change the way subsequent features are processed. All font features that 'slip through' this stage are saved in the `\zf@font@feat` macro for future processing.

```
\newcommand*\zf@fontspec[2]{%
  \begingroup
  \zf@init
  \edef\zf@fontname{#2}%
  \let\zf@family@long\zf@fontname
  \setkeys*[zf]{prepreparse}{#1}%
  \let\zf@up\zf@fontname
```

```
\edef\@tempa{\noexpand\setkeys*[zf]{preparse}{\XKV@rm}}\@tempa
\let\zf@fontname\zf@up
\let\zf@font@feat\XKV@rm
```

Now check if the font is to be rendered with ATSUI or ICU. This will either be automatic (based on the font type), or specified by the user via a font feature. If automatic, the `\zf@suffix` macro will still be empty (other suffices that could be added will be later in the feature processing), and if it is indeed still empty, assign it a value so that the other weights of the font are specifically loaded with the same renderer. This fixes a bug in v1.10 for a mishmash of Lucida fonts.

```
\font\zf@basefont="\zf@font@wrap\zf@fontname\zf@suffix" at \f@size pt
\unless\ifzf@icu
  \zf@set@font@type
\fi
\ifx\zf@suffix\@empty
  \ifzf@atsui
    \def\zf@suffix{/AAT}%
  \else
    \ifzf@icu
      \def\zf@suffix{/ICU}%
    \fi
  \fi
  \font\zf@basefont="\zf@font@wrap\zf@fontname\zf@suffix" at \f@size pt
\fi
```

Now convert the remaining 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.

In the future, this will be replaced by a dedicated `makefamily` `xkeyval` `\setkeys` declaration. Probably.

```
\zf@firsttimetrue
  \zf@get@feature@requests{\zf@font@feat}%
\zf@firsttimefalse
```

Now we have a unique (in fact, too unique!) string that contains the family name and every option in abbreviated form. This is used with a counter to create a simple NFSS family name for the font we're selecting.

The font name is fully expanded, in case it's defined in terms of macros, before having its spaces zapped.

```
\unless\ifcsname zf@UID@\zf@family@long\endcsname
  \edef\@tempa{#2}%
  \ifcsname c@zf@famc@\@tempa\endcsname
    \expandafter\stepcounter\else
    \expandafter\newcounter\fi
      {zf@famc@\@tempa}%
  \gdef@cx{zf@UID@\zf@family@long}{%
    \expandafter\expandafter\expandafter
```

```
      \zap@space\expandafter\@tempa\space\@empty
    (\expandafter\the\csname c@zf@famc@\@tempa\endcsname)}%
\fi
\xdef\zf@family{\@nameuse{zf@UID@\zf@family@long}}%
```

Now that we have the 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, use it to create global macros to save the user string of the requested options and font name, primarily for use with `\addfontfeatures`.

```
\unless\ifcsname zf@family@fontname\zf@family\endcsname
  \zf@PackageInfo{Defining font family for '#2'
        with options [\zf@default@options #1]}%
  \gdef@cx{zf@family@fontname\zf@family}{\zf@fontname}%
  \gdef@cx{zf@family@options\zf@family}{\zf@default@options #1}%
  \gdef@cx{zf@family@fontdef\zf@family}
    {\zf@fontname\zf@suffix:\zf@pre@ff\zf@ff}%
```

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: (any shape-specific features are appended to the generic font features requested in the last argument of `\zf@make@font@shapes`.)

```
  \DeclareFontFamily{\zf@enc}{\zf@family}{}%
  \zf@make@font@shapes{\zf@fontname}
    {\mddefault}{\updefault}{\zf@font@feat\zf@up@feat}%
```

Secondly, bold. Again, the extra bold options defined with `BoldFeatures` are appended to the generic font features. 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.

```
  \unless\ifzf@nobf
    \ifx\zf@bf\@empty
      \zf@make@font@shapes[\zf@fontname]{/B}
        {\bfdefault}{\updefault}{\zf@font@feat\zf@bf@feat}%
    \else
      \zf@make@font@shapes{\zf@bf}
        {\bfdefault}{\updefault}{\zf@font@feat\zf@bf@feat}%
    \fi
  \fi
```

And italic in the same way:

```
  \unless\ifzf@noit
    \ifx\zf@it\@empty
      \zf@make@font@shapes[\zf@fontname]{/I}
        {\mddefault}{\itdefault}{\zf@font@feat\zf@it@feat}%
    \else
```

```
      \zf@make@font@shapes{\zf@it}
        {\mddefault}{\itdefault}{\zf@font@feat\zf@it@feat}%
    \fi
  \fi
```

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.

```
  \@tempswatrue
  \ifzf@nobf\@tempswafalse\fi
  \ifzf@noit\@tempswafalse\fi
  \if@tempswa
    \ifx\zf@bfit\@empty
      \ifx\zf@bf\@empty
        \ifx\zf@it\@empty
          \zf@make@font@shapes[\zf@fontname]{/BI}
            {\bfdefault}{\itdefault}{\zf@font@feat\zf@bfit@feat}%
        \else
          \zf@make@font@shapes[\zf@it]{/B}
            {\bfdefault}{\itdefault}{\zf@font@feat\zf@bfit@feat}%
        \fi
      \else
        \zf@make@font@shapes[\zf@bf]{/I}
          {\bfdefault}{\itdefault}{\zf@font@feat\zf@bfit@feat}%
      \fi
    \else
      \zf@make@font@shapes{\zf@bfit}
        {\bfdefault}{\itdefault}{\zf@font@feat\zf@bfit@feat}%
    \fi
  \fi
\fi
\endgroup}
```

8.6.1 Fonts

`\zf@set@font@type` This macro sets `\zf@atsui` or `\zf@icu` or `\zf@mm` booleans accordingly depending if the font in `\zf@basefont` is an AAT font or an OpenType font or a font with feature axes (either AAT or Multiple Master), respectively.

```
\newcommand*\zf@set@font@type{%
  \zf@tfmfalse \zf@atsuifalse \zf@icufalse \zf@mmfalse
  \ifcase\XeTeXfonttype\zf@basefont
    \zf@tfm
  \or
    \zf@atsuitrue
    \ifnum\XeTeXcountvariations\zf@basefont > 0
      \zf@mmtrue
    \fi
  \or
    \zf@icutrue
  \fi}
```

`\zf@make@font@shapes [#1]`: Font name prefix
`#2` : Font name
`#3` : Font series
`#4` : Font shape
`#5` : Font features

This macro eventually uses `\DeclareFontShape` to define the font shape in question.

The optional first argument is used when making the font shapes for bold, italic, and bold italic fonts using XeTeX's auto-recognition with `#2` as `/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 and the font shape is not created for the NFSS.

```
\newcommand*\zf@make@font@shapes[5][]{%
  \begingroup
    \edef\@tempa{#1}%
    \unless\ifx\@tempa\@empty
      \font\@tempfonta="\zf@font@wrap{#1}\zf@suffix" at \f@size pt
      \edef\@tempa{\fontname\@tempfonta}%
    \fi
    \font\@tempfontb="\zf@font@wrap{#1#2}\zf@suffix" at \f@size pt
    \edef\@tempb{\fontname\@tempfontb}%
    \ifx\@tempa\@tempb
      \zf@PackageInfo{Could not resolve font #1#2 (it might not exist)}%
    \else
      \edef\zf@fontname{#1#2}%
      \let\zf@basefont\@tempfontb
      \zf@DeclareFontShape{#3}{#4}{#5}%
```

Next, the small caps are defined. `\zf@make@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 8.8 on page 61 for the code that enables this usage.

```
      \ifx\zf@sc\@empty
        \unless\ifzf@nosc
          \zf@make@smallcaps
          \unless\ifx\zf@smallcaps\@empty
            \zf@DeclareFontShape[\zf@smallcaps]{#3}
            {\ifx#4\itdefault\sidefault\else\scdefault\fi}{#5\zf@sc@feat}%
          \fi
        \fi
      \else
        \edef\zf@fontname{\zf@sc}%
        \zf@DeclareFontShape{#3}
          {\ifx#4\itdefault\sidefault\else\scdefault\fi}{#5\zf@sc@feat}%
      \fi
    \fi
  \endgroup}
```

Note that the test for italics to choose the \sidefault shape only works while \zf@fontspec passes single tokens to this macro…

\zf@DeclareFontShape [#1]: Raw appended font feature
#2 : Font series
#3 : Font shape
#4 : Font features

Wrapper for \DeclareFontShape.

```
\newcommand\zf@DeclareFontShape[4][]{%
  \ifx\zf@size@feat\@empty
    \zf@get@feature@requests{#4}%
    \edef\zf@font@str{<->\zf@scale"\zf@font@wrap\zf@fontname\zf@suffix:%
      \zf@pre@ff\zf@ff#1"}%
  \else
```

Default code, above, sets things up for no optical size fonts or features. On the other hand, loop through SizeFeatures arguments, which are of the form

SizeFeatures={{<one>},{<two>},{<three>}}.

```
    \@for\zf@this@size:=\zf@size@feat\do{%
      \let\zf@size\@empty
      \let\zf@size@fnt\zf@fontname
      \edef\@tempa{\noexpand
        \setkeys*[zf]{sizing}{\expandafter\@firstofone\zf@this@size}}%
      \@tempa
      \ifx\zf@size\@empty\zf@PackageError
          {Size information must be supplied}
          {For example, SizeFeatures={Size={8-12},...},...}%
      \fi
      \zf@get@feature@requests{#4,\XKV@rm}%
      \edef\zf@font@str{\zf@font@str <\zf@size>%
        \zf@scale"\zf@size@fnt\zf@suffix:\zf@pre@ff\zf@ff#1"}}%
  \fi
```

And finally the actual font shape declaration using \zf@font@str defined above. \zf@adjust is defined in various places to deal with things like the hyphenation character and interword spacing.

```
  \edef\@tempa{\noexpand
    \DeclareFontShape{\zf@enc}{\zf@family}{#2}{#3}
      {\zf@font@str}{\zf@adjust}}%
  \@tempa
```

This extra stuff for the slanted shape substitution is a little bit awkward, but I'd rather have it here than break out yet another macro. Alternatively, one day I might just redefine \slshape. Why not, eh?

```
  \edef\@tempa{#3}%
  \edef\@tempb{\itdefault}%
  \ifx\@tempa\@tempb
    \edef\@tempa{\noexpand
      \DeclareFontShape{\zf@enc}{\zf@family}{#2}{\sldefault}
        {<->ssub*\zf@family/#2/\itdefault}{\zf@adjust}}%
    \@tempa
  \fi}
```

`\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
    \xdef\zf@family@long{\zf@family@long#1}%
  \fi}
```

8.6.2 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[1]{%
  \let\zf@ff     \@empty
  \let\zf@scale  \@empty
  \let\zf@adjust \@empty
  \edef\@tempa{\noexpand\setkeys[zf]{options}{\zf@default@options#1}}%
  \@tempa}
```

`\zf@init` This functionality has been removed from `\zf@get@feature@requests` because it's no longer the first thing that can affect these things.

```
\newcommand*\zf@init{%
  \zf@icufalse
  \let\zf@pre@ff      \@empty
  \let\zf@font@feat   \@empty
  \let\zf@font@str    \@empty
  \let\zf@font@wrap   \@firstofone
  \let\zf@suffix      \@empty
  \let\zf@bf          \@empty
  \let\zf@it          \@empty
  \let\zf@bfit        \@empty
  \let\zf@sc          \@empty
  \let\zf@up@feat     \@empty
  \let\zf@bf@feat     \@empty
  \let\zf@it@feat     \@empty
  \let\zf@bfit@feat   \@empty
  \let\zf@sc@feat     \@empty
  \let\zf@size        \@empty
  \let\zf@size@feat   \@empty
  \let\zf@size@fnt    \@empty
  \c@zf@script 1818326126\relax
  \def\zf@script@name{Latin}%
  \c@zf@language 0\relax
  \def\zf@language@name{Default}%
}
```

`\zf@make@smallcaps` This macro checks if the font contains small caps, and if so creates the string for accessing them in `\zf@smallcaps`.

```
\newcommand*\zf@make@smallcaps{%
  \let\zf@smallcaps\@empty
  \ifzf@atsui
    \zf@make@aat@feature@string{3}{3}%
    \unless\ifx\@tempa\@empty
      \edef\zf@smallcaps{\@tempa;}%
    \fi
  \fi
  \ifzf@icu
    \zf@check@ot@feat{+smcp}%
    \if@tempswa
      \edef\zf@smallcaps{+smcp,}%
    \fi
  \fi}
```

`\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]{%
  \unless\ifzf@firsttime
    \xdef\zf@ff{\zf@ff #1\ifzf@icu,\else;\fi}%
  \fi}
```

`\zf@make@feature` This macro is called by each feature key selected, and runs according to which type of font is selected.

```
\newcommand*\zf@make@feature[3]{%
```

For AAT fonts:

```
  \ifzf@atsui
    \def\@tempa{#1}%
    \ifx\@tempa\@empty
      \zf@PackageWarning{%
        '\XKV@tfam=\XKV@tkey' feature not supported
         for AAT font \fontname\zf@basefont}%
    \else
      \zf@make@aat@feature@string{#1}{#2}%
      \ifx\@tempa\@empty
        \zf@PackageWarning{%
          AAT feature '\XKV@tfam=\XKV@tkey' (#1,#2) not available\zf@nl
          in font \fontname\zf@basefont}%
      \else
        \zf@update@family{+#1,#2}%
        \zf@update@ff\@tempa
      \fi
    \fi
  \fi
```

For OpenType fonts:

```
  \ifzf@icu
    \edef\@tempa{#3}%
    \ifx\@tempa\@empty
      \zf@PackageWarning{%
```

```
        '\XKV@tfam=\XKV@tkey' feature not supported
        for ICU font \fontname\zf@basefont}%
    \else
      \expandafter\zf@check@ot@feat\expandafter{\@tempa}%
      \if@tempswa
        \zf@update@family{#3}%
        \zf@update@ff{#3}%
      \else
        \zf@PackageWarning{%
          OpenType feature '\XKV@tfam=\XKV@tkey' (#3)
          not available\zf@nl
          for font \fontname\zf@basefont, \zf@nl
          with script '\zf@script@name',\zf@nl
          and language '\zf@language@name'.}%
      \fi
    \fi
  \fi}
```

`\zf@define@font@feature` `\zf@define@feature@option` These macros are used in order to simplify font feature definition later on.

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

`\keyval@alias@key` This macro maps one xkeyval key to another.

```
\newcommand*\keyval@alias@key[4][KV]{%
  \let@cc{#1@#2@#4}{#1@#2@#3}%
  \let@cc{#1@#2@#4@default}{#1@#2@#3@default}}
```

`\multi@alias@key` This macro iterates through families to map one key to another, regardless of which family it's contained within.

```
\newcommand*\multi@alias@key[2]{%
  \key@ifundefined[zf]{preparse}{#1}
    {\key@ifundefined[zf]{options}{#1}
       {\zf@PackageError{The feature #1 doesn't appear to be defined}
       {It looks like you're trying to rename a feature that doesn't exist.}}
       {\keyval@alias@key[zf]{options}{#1}{#2}}}
    {\keyval@alias@key[zf]{preparse}{#1}{#2}}}
```

`\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 (see page 41).

```
\newcommand*\zf@make@aat@feature@string[2]{%
  \edef\@tempa{\XeTeXfeaturename\zf@basefont #1}%
  \unless\ifx\@tempa\@empty
```

For exclusive selectors, it's easy; just grab the string:

```
    \ifnum\XeTeXisexclusivefeature\zf@basefont #1>0
      \edef\@tempb{\XeTeXselectorname\zf@basefont #1 #2}%
```

For *non*-exclusive selectors, it's a little more complex. If the selector is even, it corresponds to switching the feature on:

```
    \else
      \unless\ifodd #2
        \edef\@tempb{\XeTeXselectorname\zf@basefont #1 #2}%
```

If the selector is *odd*, it corresponds to switching the feature off. But XeTeX doesn't return a selector string for this number, since the feature is defined for the 'switching on' value. So we need to check the selector of the previous number, and then prefix the feature string with `!` to denote the switch.

```
      \else
       \edef\@tempb{\XeTeXselectorname\zf@basefont #1 \numexpr#2-1\relax}%
        \unless\ifx\@tempb\@empty
          \edef\@tempb{!\@tempb}%
        \fi
      \fi
    \fi
```

Finally, save out the complete feature string in `\@tempa`. If the selector doesn't exist, re-initialise the feature string to empty.

```
    \unless\ifx\@tempb\@empty
      \edef\@tempa{\@tempa=\@tempb}%
    \else
      \let\@tempa\@empty
    \fi
  \fi}
```

`\zf@iv@strnum` `\zf@v@strnum`

This macro takes a four character string and converts it to the numerical representation required for XeTeX OpenType script/language/feature purposes. The output is stored in `\@tempcnta`.

The reason it's ugly is because the input can be of the form of any of these: '`abcd`', '`abc`', '`abc `', '`ab`', '`ab  `', *etc*. (It is assumed the first two chars are *always* not spaces.) So this macro reads in the string, delimited by a space; this input is padded with `\@empty`s and anything beyond four chars is snipped. The `\@empty`s then are used to reconstruct the spaces in the string to number calculation.

The variant `\zf@v@strnum` is used when looking at features, which are passed around with prepended plus and minus signs (*e.g.*, `+liga`, `-dlig`); it simply strips off the first char of the input before calling the normal `\zf@iv@strnum`.

It's probable that all OpenType features *are* in fact four characters long, but not impossible that they aren't. So I'll leave the less efficient parsing stage in there even though it's not strictly necessary for now.

```
\newcommand\zf@iv@strnum[1]{%
  \zf@iv@strnum@i#1 \@nil}
\def\zf@iv@strnum@i#1 \@nil{%
  \zf@iv@strnum@ii#1\@empty\@empty\@nil}
\def\zf@iv@strnum@ii#1#2#3#4#5\@nil{%
  \@tempcnta\z@
  \@tempcntb`#1\relax
  \multiply\@tempcntb"1000000\advance\@tempcnta\@tempcntb
  \@tempcntb`#2
```

```
  \multiply\@tempcntb"10000\advance\@tempcnta\@tempcntb
  \expandafter\@tempcntb\ifx\@empty#332\else`#3\fi
  \multiply\@tempcntb"100\advance\@tempcnta\@tempcntb
  \expandafter\@tempcntb\ifx\@empty#432\else`#4\fi
  \advance\@tempcnta\@tempcntb}
\newcommand\zf@v@strnum[1]{%
  \expandafter\zf@iv@strnum@i\@gobble#1 \@nil}
```

TODO: convert to `\numexpr`

`\zf@check@ot@script` This macro takes an OpenType script tag and checks if it exists in the current font. The output boolean is `\@tempswatrue`. `\@tempcnta` is used to store the number corresponding to the script tag string.

```
\newcommand\zf@check@ot@script[1]{%
  \zf@iv@strnum{#1}%
  \@tempcntb\XeTeXOTcountscripts\zf@basefont
  \c@zf@index\z@ \@tempswafalse
  \loop\ifnum\c@zf@index<\@tempcntb
    \ifnum\XeTeXOTscripttag\zf@basefont\c@zf@index=\@tempcnta
      \@tempswatrue
      \c@zf@index\@tempcntb
    \else
      \advance\c@zf@index\@ne
    \fi
  \repeat}
```

`\zf@check@ot@lang` This macro takes an OpenType language tag and checks if it exists in the current font/script. The output boolean is `\@tempswatrue`. `\@tempcnta` is used to store the number corresponding to the language tag string. The script used is whatever's held in `\c@zf@script`. By default, that's the number corresponding to '`latn`'.

```
\newcommand\zf@check@ot@lang[1]{%
  \zf@iv@strnum{#1}%
  \@tempcntb\XeTeXOTcountlanguages\zf@basefont\c@zf@script
  \c@zf@index\z@ \@tempswafalse
  \loop\ifnum\c@zf@index<\@tempcntb
   \ifnum\XeTeXOTlanguagetag\zf@basefont\c@zf@script\c@zf@index=\@tempcnta
      \@tempswatrue
      \c@zf@index\@tempcntb
    \else
      \advance\c@zf@index\@ne
    \fi
  \repeat}
```

`\zf@check@ot@feat` This macro takes an OpenType feature tag and checks if it exists in the current font/script/language. The output boolean is `\@tempswa`. `\@tempcnta` is used to store the number corresponding to the feature tag string. The script used is whatever's held in `\c@zf@script`. By default, that's the number corresponding to '`latn`'. The language used is `\c@zf@language`, by default `0`, the 'default language'.

```
\newcommand*\zf@check@ot@feat[1]{%
  \@tempcntb\XeTeXOTcountfeatures\zf@basefont\c@zf@script\c@zf@language
  \zf@v@strnum{#1}%
```

```
\c@zf@index\z@ \@tempswafalse
\loop\ifnum\c@zf@index<\@tempcntb
  \ifnum\XeTeXOTfeaturetag\zf@basefont\c@zf@script\c@zf@language
        \c@zf@index=\@tempcnta
    \@tempswatrue
    \c@zf@index\@tempcntb
  \else
    \advance\c@zf@index\@ne
  \fi
\repeat}
```

8.7 keyval definitions

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

8.7.1 Pre-parsed features

These features are extracted from the font feature list before all others, using xkeyval's \setkeys*.

`ExternalLocation` For fonts that aren't installed in the system. If no argument is given, the font is located with kpsewhich; it's either in the current directory or the TeX tree. Otherwise, the argument given defines the file path of the font.

```
\define@key[zf]{prepreparse}{ExternalLocation}[]{%
  \zf@icutrue
  \zf@nobftrue\zf@noittrue
  \gdef\zf@font@wrap##1{[#1##1]}}
```

`Renderer` This feature must be processed before all others (the other font shape and features options are also pre-parsed for convenience) because the renderer determines the format of the features and even whether certain features are available.

```
\define@choicekey[zf]{preparse}{Renderer}{AAT,ICU}{%
  \edef\zf@suffix{\zf@suffix/#1}%
  \font\zf@basefont="\zf@font@wrap\zf@fontname\zf@suffix" at \f@size pt
  \edef\zf@family@long{\zf@family@long +rend:#1}}
```

OpenType script/language See later for the resolutions from fontspec features to OpenType definitions.

```
\define@key[zf]{preparse}{Script}{%
  \zf@icutrue
  \edef\zf@suffix{\zf@suffix/ICU}%
  \font\zf@basefont="\zf@font@wrap\zf@fontname\zf@suffix" at \f@size pt
  \edef\zf@family@long{\zf@family@long +script:#1}%
  {\setkeys[zf@feat]{Script}{#1}}}
```

Exactly the same:

```
\define@key[zf]{preparse}{Language}{%
  \zf@icutrue
  \edef\zf@suffix{\zf@suffix/ICU}%
  \font\zf@basefont="\zf@font@wrap\zf@fontname\zf@suffix" at \f@size pt
  \edef\zf@family@long{\zf@family@long +language:#1}%
  {\setkeys[zf@feat]{Lang}{#1}}}
```

8.7.2 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.

Fonts Upright:

```
\define@key[zf]{preparse}{UprightFont}{%
  \edef\@tempa{#1}%
  \zf@partial@fontname#1\@nil=\zf@up
  \edef\zf@family@long{\zf@family@long up:#1}}
```

Bold:

```
\define@key[zf]{preparse}{BoldFont}{%
  \edef\@tempa{#1}%
  \ifx\@tempa\@empty
    \zf@nobftrue
    \edef\zf@family@long{\zf@family@long nobf}%
  \else
    \zf@nobffalse
    \zf@partial@fontname#1\@nil=\zf@bf
    \edef\zf@family@long{\zf@family@long bf:#1}%
  \fi}
```

Same for italic:

```
\define@key[zf]{preparse}{ItalicFont}{%
  \edef\@tempa{#1}%
  \ifx\@tempa\@empty
    \zf@noittrue
    \edef\zf@family@long{\zf@family@long noit}%
  \else
    \zf@noitfalse
    \zf@partial@fontname#1\@nil=\zf@it
    \edef\zf@family@long{\zf@family@long it:#1}%
  \fi}
```

Simpler for bold+italic:

```
\define@key[zf]{preparse}{BoldItalicFont}{%
  \zf@partial@fontname#1\@nil=\zf@bfit
  \edef\zf@family@long{\zf@family@long bfit:#1}}
```

Small caps isn't pre-parsed because it can vary with others above:

```
\define@key[zf]{options}{SmallCapsFont}{%
```

```
  \edef\@tempa{#1}%
  \ifx\@tempa\@empty
    \zf@nosctrue
    \edef\zf@family@long{\zf@family@long nosc}%
  \else
    \zf@noscfalse
    \zf@partial@fontname#1\@nil=\zf@sc
    \zf@update@family{sc:\zap@space #1 \@empty}%
  \fi}
```

`\zf@partial@fontname` This macro takes the first token of its input and ends up defining #3 to the name of the font depending if it's been specified in full ("`Baskerville Semibold`") or in abbreviation ("`* Semibold`").

This could be done more flexibly by making * active; I'll change it later if I need to.

```
\def\zf@partial@fontname#1#2\@nil=#3{%
  \if#1*\relax
    \edef#3{\zf@fontname#2}%
  \else
    \edef#3{#1#2}%
  \fi}
```

Features

```
\define@key[zf]{preparse}{UprightFeatures}{%
  \def\zf@up@feat{,#1}%
  \edef\zf@family@long{\zf@family@long rmfeat:#1}}
\define@key[zf]{preparse}{BoldFeatures}{%
  \def\zf@bf@feat{,#1}%
  \edef\zf@family@long{\zf@family@long bffeat:#1}}
\define@key[zf]{preparse}{ItalicFeatures}{%
  \def\zf@it@feat{,#1}%
  \edef\zf@family@long{\zf@family@long itfeat:#1}}
\define@key[zf]{preparse}{BoldItalicFeatures}{%
  \def\zf@bfit@feat{,#1}%
  \edef\zf@family@long{\zf@family@long bfitfeat:#1}}
```

Note that small caps features can vary by shape, so these in fact *aren't* pre-parsed.

```
\define@key[zf]{options}{SmallCapsFeatures}{%
  \unless\ifzf@firsttime\def\zf@sc@feat{,#1}\fi
  \zf@update@family{scfeat:\zap@space #1 \@empty}}
```

paragraphFeatures varying by size TODO: sizezfeatures and italicfont (etc) don't play nice

```
\define@key[zf]{preparse}{SizeFeatures}{%
  \unless\ifzf@firsttime\def\zf@size@feat{#1}\fi
  \zf@update@family{sizefeat:\zap@space #1 \@empty}}

\define@key[zf]{sizing}{Size}{\def\zf@size{#1}}
\define@key[zf]{sizing}{Font}{\def\zf@size@fnt{#1}}
```

8.7.3 Font-independent features

These features can be applied to any font.

Scale If the input isn't one of the pre-defined string options, then it's gotta be numerical. `\zf@calc@scale` does all the work in the auto-scaling cases.

```
\define@key[zf]{options}{Scale}{%
  \edef\@tempa{#1}%
  \edef\@tempb{MatchLowercase}%
  \ifx\@tempa\@tempb
    \zf@calc@scale{5}%
  \else
    \edef\@tempb{MatchUppercase}%
    \ifx\@tempa\@tempb
      \zf@calc@scale{8}%
    \else
      \edef\zf@scale{#1}%
    \fi
  \fi
  \zf@update@family{+scale:\zf@scale}%
  \edef\zf@scale{s*[\zf@scale]}}
```

`\zf@calc@scale` This macro calculates the amount of scaling between the default roman font and the (default shape of) the font being selected such that the font dimension that is input is equal for both. The only font dimensions that justify this are 5 (lowercase height) and 8 (uppercase height in XeTeX).

This script is executed for every extra shape, which seems wasteful, but allows alternate italic shapes from a separate font, say, to be loaded and to be auto-scaled correctly. Even if this would be ugly.

```
\newcommand\zf@calc@scale[1]{%
  \begingroup
    \rmfamily
    \setlength\@tempdima{\fontdimen#1\font}%
    \setlength\@tempdimb{\fontdimen#1\zf@basefont}%
    \setlength\@tempdimc{1pt*\ratio{\@tempdima}{\@tempdimb}}%
    \xdef\zf@scale{\strip@pt\@tempdimc}%
    \zf@PackageInfo{\zf@fontname\space scale = \zf@scale}%
  \endgroup}
```

Inter-word space These options set the relevant `\fontdimen`s for the font being loaded.

```
\define@key[zf]{options}{WordSpace}{%
  \zf@update@family{+wordspace:#1}%
  \unless\ifzf@firsttime
    \zf@wordspace@parse#1,\zf@@ii,\zf@@iii,\zf@@
  \fi}
```

`\zf@wordspace@parse` This macro determines if the input to `WordSpace` is of the form `{X}` or `{X,Y,Z}` and executes the font scaling. If the former input, it executes `{X,X,X}`.

```
\def\zf@wordspace@parse#1,#2,#3,#4\zf@@{%
  \def\@tempa{#4}%
  \ifx\@tempa\@empty
    \setlength\@tempdima{#1\fontdimen2\zf@basefont}%
    \@tempdimb\@tempdima
    \@tempdimc\@tempdima
  \else
    \setlength\@tempdima{#1\fontdimen2\zf@basefont}%
    \setlength\@tempdimb{#2\fontdimen3\zf@basefont}%
    \setlength\@tempdimc{#3\fontdimen4\zf@basefont}%
  \fi
  \edef\zf@adjust{\zf@adjust
    \fontdimen2\font\the\@tempdima
    \fontdimen3\font\the\@tempdimb
    \fontdimen4\font\the\@tempdimc}}
```

Punctuation space Scaling factor for the nominal `\fontdimen#7`.

```
\define@key[zf]{options}{PunctuationSpace}{%
  \zf@update@family{+punctspace:#1}%
  \setlength\@tempdima{#1\fontdimen7\zf@basefont}%
  \edef\zf@adjust{\zf@adjust\fontdimen7\font\the\@tempdima}}
```

Letterspacing

```
\define@key[zf]{options}{LetterSpace}{%
  \zf@update@family{+tracking:#1}%
  \zf@update@ff{letterspace=#1}}
```

Hyphenation character This feature takes one of three arguments: 'None', ⟨*glyph*⟩, or ⟨*slot*⟩. If the input isn't the first, and it's one character, then it's the second; otherwise, it's the third.

```
\define@key[zf]{options}{HyphenChar}{%
  \zf@update@family{+hyphenchar:#1}%
  \edef\@tempa{#1}%
  \edef\@tempb{None}%
  \ifx\@tempa\@tempb
    \g@addto@macro\zf@adjust{\hyphenchar\font-1\relax}%
  \else
    \zf@check@one@char#1\zf@@
    \ifx\@tempb\@empty
      {\zf@basefont\expandafter\ifnum\expandafter\XeTeXcharglyph
            \expandafter`#1 > 0
         \g@addto@macro\zf@adjust{%
           {\expandafter\hyphenchar\expandafter
            \font\expandafter`#1}}%
      \else
        \zf@PackageError
          {\fontname\zf@basefont\space doesn't appear to have the glyph
            corresponding to #1.}
          {You can't hyphenate with a character that's not available!}
```

```
      \fi}%
    \else
      {\zf@basefont\ifnum\XeTeXcharglyph#1 > 0
        \g@addto@macro\zf@adjust{\hyphenchar\font#1\relax}%
      \else
        \zf@PackageError
          {\fontname\zf@basefont\space doesn't appear to have the glyph
            corresponding to #1.}
          {You can't hyphenate with a character that's not available!}%
      \fi}%
    \fi
  \fi}
\def\zf@check@one@char#1#2\zf@@{\def\@tempb{#2}}
```

Colour

```
\define@key[zf]{options}{Colour}{%
  \zf@update@family{+col:#1}%
  \zf@update@ff{color=#1}}
\keyval@alias@key[zf]{options}{Colour}{Color}
```

Mapping

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

8.7.4 Continuous font axes

```
\define@key[zf]{options}{Weight}{%
  \zf@update@family{+weight:#1}%
  \zf@update@ff{weight=#1}}
\define@key[zf]{options}{Width}{%
  \zf@update@family{+width:#1}%
  \zf@update@ff{width=#1}}
\define@key[zf]{options}{OpticalSize}{%
  \ifzf@icu
    \edef\zf@suffix{\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@icu\else
    \ifzf@mm\else
      \ifzf@firsttime
        \zf@PackageWarning
          {\fontname\zf@basefont\space doesn't appear
            to have an Optical Size axis}%
      \fi
    \fi
```

```
  \fi}
```

8.7.5 Font transformations

```
\define@key[zf]{options}{FakeSlant}{%
  \zf@update@family{+slant:#1}%
  \zf@update@ff{slant=#1}}
\define@key[zf]{options}{FakeStretch}{%
  \zf@update@family{+extend:#1}%
  \zf@update@ff{extend=#1}}
\define@key[zf]{options}{FakeBold}{%
  \zf@update@family{+embolden:#1}%
  \zf@update@ff{embolden=#1}}
```

8.7.6 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}{}
```

8.7.7 Letters

```
\zf@define@font@feature{Letters}
\zf@define@feature@option{Letters}{Normal}             {3}{0}{}
\zf@define@feature@option{Letters}{Uppercase}          {3}{1}{+case}
\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}{UppercaseSmallCaps} {} {} {+c2sc}
\zf@define@feature@option{Letters}{UppercasePetiteCaps}{} {} {+c2pc}
\zf@define@feature@option{Letters}{InitialCaps}        {3}{4}{}
\zf@define@feature@option{Letters}{Unicase}            {} {} {+unic}
```

8.7.8 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}
```

8.7.9 Contextuals

```
\zf@define@font@feature  {Contextuals}
\zf@define@feature@option{Contextuals}{Swash}        {} {} {+cswh}
\zf@define@feature@option{Contextuals}{NoSwash}      {} {} {-cswh}
\zf@define@feature@option{Contextuals}{Alternate}    {} {} {+calt}
\zf@define@feature@option{Contextuals}{NoAlternate}  {} {} {-calt}
\zf@define@feature@option{Contextuals}{WordInitial}  {8}{0}{+init}
\zf@define@feature@option{Contextuals}{NoWordInitial}{8}{1}{-init}
\zf@define@feature@option{Contextuals}{WordFinal}    {8}{2}{+fina}
\zf@define@feature@option{Contextuals}{NoWordFinal}  {8}{3}{-fina}
\zf@define@feature@option{Contextuals}{LineInitial}  {8}{4}{}
\zf@define@feature@option{Contextuals}{NoLineInitial}{8}{5}{}
\zf@define@feature@option{Contextuals}{LineFinal}    {8}{6}{+falt}
\zf@define@feature@option{Contextuals}{NoLineFinal}  {8}{7}{-falt}
\zf@define@feature@option{Contextuals}{Inner}        {8}{8}{+medi}
\zf@define@feature@option{Contextuals}{NoInner}      {8}{9}{-medi}
```

8.7.10 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}{}
```

8.7.11 Kerning

```
\zf@define@font@feature{Kerning}
\zf@define@feature@option{Kerning}{Uppercase}{}{}{+cpsp}
\zf@define@feature@option{Kerning}{On}       {}{}{+kern}
\zf@define@feature@option{Kerning}{Off}      {}{}{-kern}
%\zf@define@feature@option{Kerning}{Vertical}{}{}{+vkrn}
%\zf@define@feature@option{Kerning}
```

```
%    {VerticalAlternateProportional}{}{}{+vpal}
%\zf@define@feature@option{Kerning}{VerticalAlternateHalfWidth}{}{}{+vhal}
```

8.7.12 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}{Ordinal}    {10}{3}{+ordn}
\zf@define@feature@option{VerticalPosition}{Numerator}  {}  {} {+numr}
\zf@define@feature@option{VerticalPosition}{Denominator}{}  {} {+dnom}
\zf@define@feature@option{VerticalPosition}{ScientificInferior}{}{}{+sinf}
```

8.7.13 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}
```

8.7.14 Alternates and variants

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

```
\define@key[zf]{options}{Alternate}[0]{%
  \setkeys*[zf@feat]{Alternate}{#1}%
  \unless\ifx\XKV@rm\@empty
    \def\XKV@tfam{Alternate}%
    \zf@make@feature{17}{#1}{+salt=#1}%
  \fi}
```

```
\define@key[zf]{options}{Variant}{%
  \setkeys*[zf@feat]{Variant}{#1}%
  \unless\ifx\XKV@rm\@empty
    \def\XKV@tfam{Variant}%
    \zf@make@feature{18}{#1}{+ss\two@digits{#1}}%
  \fi}
```

8.7.15 Style

```
\zf@define@font@feature{Style}
\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}{Historic}      {}  {} {+hist}
\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}{}
\zf@define@feature@option{Style}{HorizontalKana}{}  {} {+hkna}
\zf@define@feature@option{Style}{VerticalKana}  {}  {} {+vkna}
```

8.7.16 CJK shape

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

8.7.17 Character width

```
\zf@define@font@feature{CharacterWidth}
\zf@define@feature@option{CharacterWidth}{Proportional}{22}{0}{+pwid}
\zf@define@feature@option{CharacterWidth}{Full}{22}{1}{+fwid}
\zf@define@feature@option{CharacterWidth}{Half}{22}{2}{+hwid}
\zf@define@feature@option{CharacterWidth}{Third}{22}{3}{+twid}
\zf@define@feature@option{CharacterWidth}{Quarter}{22}{4}{+qwid}
\zf@define@feature@option{CharacterWidth}{AlternateProportional}{22}{5}{+palt}
\zf@define@feature@option{CharacterWidth}{AlternateHalf}{22}{6}{+halt}
\zf@define@feature@option{CharacterWidth}{Default}{22}{7}{}
```

8.7.18 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}{}
```

8.7.19 Vertical

```
\zf@define@font@feature{Vertical}
\define@key[zf@feat]{Vertical}{RotatedGlyphs}[]{%
  \ifzf@icu
    \zf@make@feature{}{}{+vrt2}%
    \zf@update@family{+vert}%
    \zf@update@ff{vertical}%
  \else
    \zf@update@family{+vert}%
    \zf@update@ff{vertical}%
  \fi}
```

8.7.20 Script

```
\newfontscript{Arabic}{arab}            \newfontscript{Armenian}{armn}
\newfontscript{Balinese}{bali}          \newfontscript{Bengali}{beng}
\newfontscript{Bopomofo}{bopo}          \newfontscript{Braille}{brai}
\newfontscript{Buginese}{bugi}          \newfontscript{Buhid}{buhd}
\newfontscript{Byzantine Music}{byzm} \newfontscript{Canadian Syllabics}{cans}
\newfontscript{Cherokee}{cher}
\newfontscript{CJK Ideographic}{hani}   \newfontscript{Coptic}{copt}
\newfontscript{Cypriot Syllabary}{cprt} \newfontscript{Cyrillic}{cyrl}
\newfontscript{Default}{DFLT}           \newfontscript{Deseret}{dsrt}
\newfontscript{Devanagari}{deva}        \newfontscript{Ethiopic}{ethi}
\newfontscript{Georgian}{geor}          \newfontscript{Glagolitic}{glag}
\newfontscript{Gothic}{goth}            \newfontscript{Greek}{grek}
\newfontscript{Gujarati}{gujr}          \newfontscript{Gurmukhi}{guru}
\newfontscript{Hangul Jamo}{jamo}       \newfontscript{Hangul}{hang}
\newfontscript{Hanunoo}{hano}           \newfontscript{Hebrew}{hebr}
\newfontscript{Hiragana and Katakana}{kana}
\newfontscript{Javanese}{java}          \newfontscript{Kannada}{knda}
\newfontscript{Kharosthi}{khar}         \newfontscript{Khmer}{khmr}
\newfontscript{Lao}{lao }               \newfontscript{Latin}{latn}
\newfontscript{Limbu}{limb}             \newfontscript{Linear B}{linb}
\newfontscript{Malayalam}{mlym}         \newfontscript{Math}{math}
\newfontscript{Mongolian}{mong}
\newfontscript{Musical Symbols}{musc}   \newfontscript{Myanmar}{mymr}
\newfontscript{N'ko}{nko }              \newfontscript{Ogham}{ogam}
\newfontscript{Old Italic}{ital}
\newfontscript{Old Persian Cuneiform}{xpeo}
\newfontscript{Oriya}{orya}             \newfontscript{Osmanya}{osma}
\newfontscript{Phags-pa}{phag}          \newfontscript{Phoenician}{phnx}
\newfontscript{Runic}{runr}             \newfontscript{Shavian}{shaw}
\newfontscript{Sinhala}{sinh}
\newfontscript{Sumero-Akkadian Cuneiform}{xsux}
\newfontscript{Syloti Nagri}{sylo}      \newfontscript{Syriac}{syrc}
\newfontscript{Tagalog}{tglg}           \newfontscript{Tagbanwa}{tagb}
\newfontscript{Tai Le}{tale}            \newfontscript{Tai Lu}{talu}
\newfontscript{Tamil}{taml}             \newfontscript{Telugu}{telu}
\newfontscript{Thaana}{thaa}            \newfontscript{Thai}{thai}
\newfontscript{Tibetan}{tibt}           \newfontscript{Tifinagh}{tfng}
\newfontscript{Ugaritic Cuneiform}{ugar}\newfontscript{Yi}{yi  }
```

8.7.21 Language

```
\newfontlanguage{Abaza}{ABA}\newfontlanguage{Abkhazian}{ABK}
\newfontlanguage{Adyghe}{ADY}\newfontlanguage{Afrikaans}{AFK}
\newfontlanguage{Afar}{AFR}\newfontlanguage{Agaw}{AGW}
\newfontlanguage{Altai}{ALT}\newfontlanguage{Amharic}{AMH}
\newfontlanguage{Arabic}{ARA}\newfontlanguage{Aari}{ARI}
\newfontlanguage{Arakanese}{ARK}\newfontlanguage{Assamese}{ASM}
\newfontlanguage{Athapaskan}{ATH}\newfontlanguage{Avar}{AVR}
\newfontlanguage{Awadhi}{AWA}\newfontlanguage{Aymara}{AYM}
\newfontlanguage{Azeri}{AZE}\newfontlanguage{Badaga}{BAD}
\newfontlanguage{Baghelkhandi}{BAG}\newfontlanguage{Balkar}{BAL}
\newfontlanguage{Baule}{BAU}\newfontlanguage{Berber}{BBR}
```

```
\newfontlanguage{Bench}{BCH}\newfontlanguage{Bible Cree}{BCR}
\newfontlanguage{Belarussian}{BEL}\newfontlanguage{Bemba}{BEM}
\newfontlanguage{Bengali}{BEN}\newfontlanguage{Bulgarian}{BGR}
\newfontlanguage{Bhili}{BHI}\newfontlanguage{Bhojpuri}{BHO}
\newfontlanguage{Bikol}{BIK}\newfontlanguage{Bilen}{BIL}
\newfontlanguage{Blackfoot}{BKF}\newfontlanguage{Balochi}{BLI}
\newfontlanguage{Balante}{BLN}\newfontlanguage{Balti}{BLT}
\newfontlanguage{Bambara}{BMB}\newfontlanguage{Bamileke}{BML}
\newfontlanguage{Breton}{BRE}\newfontlanguage{Brahui}{BRH}
\newfontlanguage{Braj Bhasha}{BRI}\newfontlanguage{Burmese}{BRM}
\newfontlanguage{Bashkir}{BSH}\newfontlanguage{Beti}{BTI}
\newfontlanguage{Catalan}{CAT}\newfontlanguage{Cebuano}{CEB}
\newfontlanguage{Chechen}{CHE}\newfontlanguage{Chaha Gurage}{CHG}
\newfontlanguage{Chattisgarhi}{CHH}\newfontlanguage{Chichewa}{CHI}
\newfontlanguage{Chukchi}{CHK}\newfontlanguage{Chipewyan}{CHP}
\newfontlanguage{Cherokee}{CHR}\newfontlanguage{Chuvash}{CHU}
\newfontlanguage{Comorian}{CMR}\newfontlanguage{Coptic}{COP}
\newfontlanguage{Cree}{CRE}\newfontlanguage{Carrier}{CRR}
\newfontlanguage{Crimean Tatar}{CRT}\newfontlanguage{Church Slavonic}{CSL}
\newfontlanguage{Czech}{CSY}\newfontlanguage{Danish}{DAN}
\newfontlanguage{Dargwa}{DAR}\newfontlanguage{Woods Cree}{DCR}
\newfontlanguage{German}{DEU}\newfontlanguage{Default}{DFLT}
\newfontlanguage{Dogri}{DGR}\newfontlanguage{Divehi}{DIV}
\newfontlanguage{Djerma}{DJR}\newfontlanguage{Dangme}{DNG}
\newfontlanguage{Dinka}{DNK}\newfontlanguage{Dungan}{DUN}
\newfontlanguage{Dzongkha}{DZN}\newfontlanguage{Ebira}{EBI}
\newfontlanguage{Eastern Cree}{ECR}\newfontlanguage{Edo}{EDO}
\newfontlanguage{Efik}{EFI}\newfontlanguage{Greek}{ELL}
\newfontlanguage{English}{ENG}\newfontlanguage{Erzya}{ERZ}
\newfontlanguage{Spanish}{ESP}\newfontlanguage{Estonian}{ETI}
\newfontlanguage{Basque}{EUQ}\newfontlanguage{Evenki}{EVK}
\newfontlanguage{Even}{EVN}\newfontlanguage{Ewe}{EWE}
\newfontlanguage{French Antillean}{FAN}\newfontlanguage{Farsi}{FAR}
\newfontlanguage{Finnish}{FIN}\newfontlanguage{Fijian}{FJI}
\newfontlanguage{Flemish}{FLE}\newfontlanguage{Forest Nenets}{FNE}
\newfontlanguage{Fon}{FON}\newfontlanguage{Faroese}{FOS}
\newfontlanguage{French}{FRA}\newfontlanguage{Frisian}{FRI}
\newfontlanguage{Friulian}{FRL}\newfontlanguage{Futa}{FTA}
\newfontlanguage{Fulani}{FUL}\newfontlanguage{Ga}{GAD}
\newfontlanguage{Gaelic}{GAE}\newfontlanguage{Gagauz}{GAG}
\newfontlanguage{Galician}{GAL}\newfontlanguage{Garshuni}{GAR}
\newfontlanguage{Garhwali}{GAW}\newfontlanguage{Ge'ez}{GEZ}
\newfontlanguage{Gilyak}{GIL}\newfontlanguage{Gumuz}{GMZ}
\newfontlanguage{Gondi}{GON}\newfontlanguage{Greenlandic}{GRN}
\newfontlanguage{Garo}{GRO}\newfontlanguage{Guarani}{GUA}
\newfontlanguage{Gujarati}{GUJ}\newfontlanguage{Haitian}{HAI}
\newfontlanguage{Halam}{HAL}\newfontlanguage{Harauti}{HAR}
\newfontlanguage{Hausa}{HAU}\newfontlanguage{Hawaiin}{HAW}
\newfontlanguage{Hammer-Banna}{HBN}\newfontlanguage{Hiligaynon}{HIL}
\newfontlanguage{Hindi}{HIN}\newfontlanguage{High Mari}{HMA}
\newfontlanguage{Hindko}{HND}\newfontlanguage{Ho}{HO}
```

```
\newfontlanguage{Harari}{HRI}\newfontlanguage{Croatian}{HRV}
\newfontlanguage{Hungarian}{HUN}\newfontlanguage{Armenian}{HYE}
\newfontlanguage{Igbo}{IBO}\newfontlanguage{Ijo}{IJO}
\newfontlanguage{Ilokano}{ILO}\newfontlanguage{Indonesian}{IND}
\newfontlanguage{Ingush}{ING}\newfontlanguage{Inuktitut}{INU}
\newfontlanguage{Irish}{IRI}\newfontlanguage{Irish Traditional}{IRT}
\newfontlanguage{Icelandic}{ISL}\newfontlanguage{Inari Sami}{ISM}
\newfontlanguage{Italian}{ITA}\newfontlanguage{Hebrew}{IWR}
\newfontlanguage{Javanese}{JAV}\newfontlanguage{Yiddish}{JII}
\newfontlanguage{Japanese}{JAN}\newfontlanguage{Judezmo}{JUD}
\newfontlanguage{Jula}{JUL}\newfontlanguage{Kabardian}{KAB}
\newfontlanguage{Kachchi}{KAC}\newfontlanguage{Kalenjin}{KAL}
\newfontlanguage{Kannada}{KAN}\newfontlanguage{Karachay}{KAR}
\newfontlanguage{Georgian}{KAT}\newfontlanguage{Kazakh}{KAZ}
\newfontlanguage{Kebena}{KEB}\newfontlanguage{Khutsuri Georgian}{KGE}
\newfontlanguage{Khakass}{KHA}\newfontlanguage{Khanty-Kazim}{KHK}
\newfontlanguage{Khmer}{KHM}\newfontlanguage{Khanty-Shurishkar}{KHS}
\newfontlanguage{Khanty-Vakhi}{KHV}\newfontlanguage{Khowar}{KHW}
\newfontlanguage{Kikuyu}{KIK}\newfontlanguage{Kirghiz}{KIR}
\newfontlanguage{Kisii}{KIS}\newfontlanguage{Kokni}{KKN}
\newfontlanguage{Kalmyk}{KLM}\newfontlanguage{Kamba}{KMB}
\newfontlanguage{Kumaoni}{KMN}\newfontlanguage{Komo}{KMO}
\newfontlanguage{Komso}{KMS}\newfontlanguage{Kanuri}{KNR}
\newfontlanguage{Kodagu}{KOD}\newfontlanguage{Korean Old Hangul}{KOH}
\newfontlanguage{Konkani}{KOK}\newfontlanguage{Kikongo}{KON}
\newfontlanguage{Komi-Permyak}{KOP}\newfontlanguage{Korean}{KOR}
\newfontlanguage{Komi-Zyrian}{KOZ}\newfontlanguage{Kpelle}{KPL}
\newfontlanguage{Krio}{KRI}\newfontlanguage{Karakalpak}{KRK}
\newfontlanguage{Karelian}{KRL}\newfontlanguage{Karaim}{KRM}
\newfontlanguage{Karen}{KRN}\newfontlanguage{Koorete}{KRT}
\newfontlanguage{Kashmiri}{KSH}\newfontlanguage{Khasi}{KSI}
\newfontlanguage{Kildin Sami}{KSM}\newfontlanguage{Kui}{KUI}
\newfontlanguage{Kulvi}{KUL}\newfontlanguage{Kumyk}{KUM}
\newfontlanguage{Kurdish}{KUR}\newfontlanguage{Kurukh}{KUU}
\newfontlanguage{Kuy}{KUY}\newfontlanguage{Koryak}{KYK}
\newfontlanguage{Ladin}{LAD}\newfontlanguage{Lahuli}{LAH}
\newfontlanguage{Lak}{LAK}\newfontlanguage{Lambani}{LAM}
\newfontlanguage{Lao}{LAO}\newfontlanguage{Latin}{LAT}
\newfontlanguage{Laz}{LAZ}\newfontlanguage{L-Cree}{LCR}
\newfontlanguage{Ladakhi}{LDK}\newfontlanguage{Lezgi}{LEZ}
\newfontlanguage{Lingala}{LIN}\newfontlanguage{Low Mari}{LMA}
\newfontlanguage{Limbu}{LMB}\newfontlanguage{Lomwe}{LMW}
\newfontlanguage{Lower Sorbian}{LSB}\newfontlanguage{Lule Sami}{LSM}
\newfontlanguage{Lithuanian}{LTH}\newfontlanguage{Luba}{LUB}
\newfontlanguage{Luganda}{LUG}\newfontlanguage{Luhya}{LUH}
\newfontlanguage{Luo}{LUO}\newfontlanguage{Latvian}{LVI}
\newfontlanguage{Majang}{MAJ}\newfontlanguage{Makua}{MAK}
\newfontlanguage{Malayalam Traditional}{MAL}\newfontlanguage{Mansi}{MAN}
\newfontlanguage{Marathi}{MAR}\newfontlanguage{Marwari}{MAW}
\newfontlanguage{Mbundu}{MBN}\newfontlanguage{Manchu}{MCH}
\newfontlanguage{Moose Cree}{MCR}\newfontlanguage{Mende}{MDE}
```

```
\newfontlanguage{Me'en}{MEN}\newfontlanguage{Mizo}{MIZ}
\newfontlanguage{Macedonian}{MKD}\newfontlanguage{Male}{MLE}
\newfontlanguage{Malagasy}{MLG}\newfontlanguage{Malinke}{MLN}
\newfontlanguage{Malayalam Reformed}{MLR}\newfontlanguage{Malay}{MLY}
\newfontlanguage{Mandinka}{MND}\newfontlanguage{Mongolian}{MNG}
\newfontlanguage{Manipuri}{MNI}\newfontlanguage{Maninka}{MNK}
\newfontlanguage{Manx Gaelic}{MNX}\newfontlanguage{Moksha}{MOK}
\newfontlanguage{Moldavian}{MOL}\newfontlanguage{Mon}{MON}
\newfontlanguage{Moroccan}{MOR}\newfontlanguage{Maori}{MRI}
\newfontlanguage{Maithili}{MTH}\newfontlanguage{Maltese}{MTS}
\newfontlanguage{Mundari}{MUN}\newfontlanguage{Naga-Assamese}{NAG}
\newfontlanguage{Nanai}{NAN}\newfontlanguage{Naskapi}{NAS}
\newfontlanguage{N-Cree}{NCR}\newfontlanguage{Ndebele}{NDB}
\newfontlanguage{Ndonga}{NDG}\newfontlanguage{Nepali}{NEP}
\newfontlanguage{Newari}{NEW}\newfontlanguage{Nagari}{NGR}
\newfontlanguage{Norway House Cree}{NHC}\newfontlanguage{Nisi}{NIS}
\newfontlanguage{Niuean}{NIU}\newfontlanguage{Nkole}{NKL}
\newfontlanguage{N'ko}{NKO}\newfontlanguage{Dutch}{NLD}
\newfontlanguage{Nogai}{NOG}\newfontlanguage{Norwegian}{NOR}
\newfontlanguage{Northern Sami}{NSM}\newfontlanguage{Northern Tai}{NTA}
\newfontlanguage{Esperanto}{NTO}\newfontlanguage{Nynorsk}{NYN}
\newfontlanguage{Oji-Cree}{OCR}\newfontlanguage{Ojibway}{OJB}
\newfontlanguage{Oriya}{ORI}\newfontlanguage{Oromo}{ORO}
\newfontlanguage{Ossetian}{OSS}\newfontlanguage{Palestinian Aramaic}{PAA}
\newfontlanguage{Pali}{PAL}\newfontlanguage{Punjabi}{PAN}
\newfontlanguage{Palpa}{PAP}\newfontlanguage{Pashto}{PAS}
\newfontlanguage{Polytonic Greek}{PGR}\newfontlanguage{Pilipino}{PIL}
\newfontlanguage{Palaung}{PLG}\newfontlanguage{Polish}{PLK}
\newfontlanguage{Provencal}{PRO}\newfontlanguage{Portuguese}{PTG}
\newfontlanguage{Chin}{QIN}\newfontlanguage{Rajasthani}{RAJ}
\newfontlanguage{R-Cree}{RCR}\newfontlanguage{Russian Buriat}{RBU}
\newfontlanguage{Riang}{RIA}\newfontlanguage{Rhaeto-Romanic}{RMS}
\newfontlanguage{Romanian}{ROM}\newfontlanguage{Romany}{ROY}
\newfontlanguage{Rusyn}{RSY}\newfontlanguage{Ruanda}{RUA}
\newfontlanguage{Russian}{RUS}\newfontlanguage{Sadri}{SAD}
\newfontlanguage{Sanskrit}{SAN}\newfontlanguage{Santali}{SAT}
\newfontlanguage{Sayisi}{SAY}\newfontlanguage{Sekota}{SEK}
\newfontlanguage{Selkup}{SEL}\newfontlanguage{Sango}{SGO}
\newfontlanguage{Shan}{SHN}\newfontlanguage{Sibe}{SIB}
\newfontlanguage{Sidamo}{SID}\newfontlanguage{Silte Gurage}{SIG}
\newfontlanguage{Skolt Sami}{SKS}\newfontlanguage{Slovak}{SKY}
\newfontlanguage{Slavey}{SLA}\newfontlanguage{Slovenian}{SLV}
\newfontlanguage{Somali}{SML}\newfontlanguage{Samoan}{SMO}
\newfontlanguage{Sena}{SNA}\newfontlanguage{Sindhi}{SND}
\newfontlanguage{Sinhalese}{SNH}\newfontlanguage{Soninke}{SNK}
\newfontlanguage{Sodo Gurage}{SOG}\newfontlanguage{Sotho}{SOT}
\newfontlanguage{Albanian}{SQI}\newfontlanguage{Serbian}{SRB}
\newfontlanguage{Saraiki}{SRK}\newfontlanguage{Serer}{SRR}
\newfontlanguage{South Slavey}{SSL}\newfontlanguage{Southern Sami}{SSM}
\newfontlanguage{Suri}{SUR}\newfontlanguage{Svan}{SVA}
\newfontlanguage{Swedish}{SVE}\newfontlanguage{Swadaya Aramaic}{SWA}
```

```
\newfontlanguage{Swahili}{SWK}\newfontlanguage{Swazi}{SWZ}
\newfontlanguage{Sutu}{SXT}\newfontlanguage{Syriac}{SYR}
\newfontlanguage{Tabasaran}{TAB}\newfontlanguage{Tajiki}{TAJ}
\newfontlanguage{Tamil}{TAM}\newfontlanguage{Tatar}{TAT}
\newfontlanguage{TH-Cree}{TCR}\newfontlanguage{Telugu}{TEL}
\newfontlanguage{Tongan}{TGN}\newfontlanguage{Tigre}{TGR}
\newfontlanguage{Tigrinya}{TGY}\newfontlanguage{Thai}{THA}
\newfontlanguage{Tahitian}{THT}\newfontlanguage{Tibetan}{TIB}
\newfontlanguage{Turkmen}{TKM}\newfontlanguage{Temne}{TMN}
\newfontlanguage{Tswana}{TNA}\newfontlanguage{Tundra Nenets}{TNE}
\newfontlanguage{Tonga}{TNG}\newfontlanguage{Todo}{TOD}
\newfontlanguage{Tsonga}{TSG}\newfontlanguage{Turoyo Aramaic}{TUA}
\newfontlanguage{Tulu}{TUL}\newfontlanguage{Tuvin}{TUV}
\newfontlanguage{Twi}{TWI}\newfontlanguage{Udmurt}{UDM}
\newfontlanguage{Ukrainian}{UKR}\newfontlanguage{Urdu}{URD}
\newfontlanguage{Upper Sorbian}{USB}\newfontlanguage{Uyghur}{UYG}
\newfontlanguage{Uzbek}{UZB}\newfontlanguage{Venda}{VEN}
\newfontlanguage{Vietnamese}{VIT}\newfontlanguage{Wa}{WA}
\newfontlanguage{Wagdi}{WAG}\newfontlanguage{West-Cree}{WCR}
\newfontlanguage{Welsh}{WEL}\newfontlanguage{Wolof}{WLF}
\newfontlanguage{Tai Lue}{XBD}\newfontlanguage{Xhosa}{XHS}
\newfontlanguage{Yakut}{YAK}\newfontlanguage{Yoruba}{YBA}
\newfontlanguage{Y-Cree}{YCR}\newfontlanguage{Yi Classic}{YIC}
\newfontlanguage{Yi Modern}{YIM}\newfontlanguage{Chinese Hong Kong}{ZHH}
\newfontlanguage{Chinese Phonetic}{ZHP}\newfontlanguage{Chinese Simplified}{ZHS}
\newfontlanguage{Chinese Traditional}{ZHT}\newfontlanguage{Zande}{ZND}
\newfontlanguage{Zulu}{ZUL}
```

Turkish Turns out that many fonts use 'TUR' as their Turkish language tag rather than the specified 'TRK'. So we check for both:

```
\define@key[zf@feat]{Lang}{Turkish}[]{%
  \zf@check@ot@lang{TRK}%
  \if@tempswa
    \c@zf@language\@tempcnta\relax
    \xdef\zf@language@name{Turkish}%
    \xdef\zf@family@long{\zf@family@long+lang=Turkish}%
    \xdef\zf@pre@ff{\zf@pre@ff language=TRK,}%
  \else
    \zf@check@ot@lang{TUR}%
    \if@tempswa
      \c@zf@language\@tempcnta\relax
      \xdef\zf@language@name{Turkish}%
      \xdef\zf@family@long{\zf@family@long+lang=Turkish}%
      \xdef\zf@pre@ff{\zf@pre@ff language=TUR,}%
    \else
      \zf@PackageWarning{Language '#1' not available\zf@nl
         for font \fontname\zf@basefont\zf@nl
         with script '\zf@script@name'.}%
    \fi
  \fi}
```

8.7.22 Raw feature string

This allows savvy XeTeX-ers to input font features manually if they have already memorised the OpenType abbreviations and don't mind not having error checking.

```
\define@key[zf]{options}{RawFeature}{%
  \zf@update@family{+Raw:#1}%
  \zf@update@ff{#1}}
```

8.8 Italic small caps

The following code for utilising italic small caps sensibly is inspired from Philip Lehman's *The 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
    \ifcsname\f@encoding/\f@family/\f@series/#3\endcsname
      \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{\slshape}{%
  \not@math@alphabet\slshape\relax
  \zf@merge@shape\sldefault\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}
```

\em
\emph

Redefinitions moved to the xltxtra package.

8.9 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 \setmainfont and friends was run.

\zf@math

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 XeTeX 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.

```
\@ifpackageloaded{euler}{\zf@package@euler@loadedtrue}
                        {\zf@package@euler@loadedfalse}
\def\zf@math{%
  \let\zf@font@warning\@font@warning
  \let\@font@warning\@font@info
  \@ifpackageloaded{euler}{%
    \ifzf@package@euler@loaded
      \zf@math@eulertrue
    \else
      \zf@PackageError{The euler package must be loaded BEFORE fontspec}
        {fontspec only overwrites euler's attempt to  ^^J
         define the maths text fonts if fontspec is   ^^J
         loaded after euler. Type <return> to proceed ^^J
         with incorrect \protect\mathit, \protect\mathbf, etc}
    \fi}{}
  \@ifpackageloaded{lucbmath}{\zf@math@lucidatrue}{}
  \@ifpackageloaded{lucidabr}{\zf@math@lucidatrue}{}
  \@ifpackageloaded{lucimatx}{\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}{19}
\DeclareMathAccent{\grave}   {\mathalpha}{legacymaths}{18}
\DeclareMathAccent{\ddot}    {\mathalpha}{legacymaths}{127}
\DeclareMathAccent{\tilde}   {\mathalpha}{legacymaths}{126}
\DeclareMathAccent{\bar}     {\mathalpha}{legacymaths}{22}
\DeclareMathAccent{\breve}   {\mathalpha}{legacymaths}{21}
\DeclareMathAccent{\check}   {\mathalpha}{legacymaths}{20}
\DeclareMathAccent{\hat}    {\mathalpha}{legacymaths}{94} % too bad, euler
\DeclareMathAccent{\dot}     {\mathalpha}{legacymaths}{95}
\DeclareMathAccent{\mathring}{\mathalpha}{legacymaths}{23}
```

`\colon`**: what's going on?** Okay, so `:` and `\colon` in maths mode are defined in a few places, so I need to work out what does what. Respectively, we have:

```
% fontmath.ltx:
\DeclareMathSymbol{\colon}{\mathpunct}{operators}{"3A}
\DeclareMathSymbol{:}{\mathrel}{operators}{"3A}

% amsmath.sty:
\renewcommand{\colon}{\nobreak\mskip2mu\mathpunct{}\nonscript
 \mkern-\thinmuskip{:}\mskip6muplus1mu\relax}

% euler.sty:
\DeclareMathSymbol{:}\mathrel  {EulerFraktur}{"3A}

% lucbmath.sty:
\DeclareMathSymbol{\@tempb}{\mathpunct}{operators}{58}
\ifx\colon\@tempb
  \DeclareMathSymbol{\colon}{\mathpunct}{operators}{58}
\fi
\DeclareMathSymbol{:}{\mathrel}{operators}{58}
```

($3A_{16} = 58_{10}$) So I think, based on this summary, that it is fair to tell fontspec to 'replace' the operators font with legacymaths for this symbol, except when amsmath is loaded since we want to keep its definition.

```
\begingroup
  \mathchardef\@tempa="603A %
  \let\next\egroup
  \ifx\colon\@tempa
    \DeclareMathSymbol{\colon}{\mathpunct}{legacymaths}{58}
  \fi
\endgroup
```

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

```
\ifzf@math@euler\else
  \DeclareMathSymbol{!}{\mathclose}{legacymaths}{33}
  \DeclareMathSymbol{:}{\mathrel}  {legacymaths}{58}
  \DeclareMathSymbol{;}{\mathpunct}{legacymaths}{59}
  \DeclareMathSymbol{?}{\mathclose}{legacymaths}{63}
```

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.

```
  \ifzf@math@lucida\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}{0}
    \DeclareMathSymbol{\Delta}{\mathalpha}{legacymaths}{1}
    \DeclareMathSymbol{\Theta}{\mathalpha}{legacymaths}{2}
    \DeclareMathSymbol{\Lambda}{\mathalpha}{legacymaths}{3}
    \DeclareMathSymbol{\Xi}{\mathalpha}{legacymaths}{4}
    \DeclareMathSymbol{\Pi}{\mathalpha}{legacymaths}{5}
    \DeclareMathSymbol{\Sigma}{\mathalpha}{legacymaths}{6}
    \DeclareMathSymbol{\Upsilon}{\mathalpha}{legacymaths}{7}
    \DeclareMathSymbol{\Phi}{\mathalpha}{legacymaths}{8}
    \DeclareMathSymbol{\Psi}{\mathalpha}{legacymaths}{9}
    \DeclareMathSymbol{\Omega}{\mathalpha}{legacymaths}{10}
    \DeclareMathSymbol{+}{\mathbin}{legacymaths}{43}
    \DeclareMathSymbol{=}{\mathrel}{legacymaths}{61}
   \DeclareMathDelimiter{(}{\mathopen} {legacymaths}{40}{largesymbols}{0}
   \DeclareMathDelimiter{)}{\mathclose}{legacymaths}{41}{largesymbols}{1}
   \DeclareMathDelimiter{[}{\mathopen} {legacymaths}{91}{largesymbols}{2}
   \DeclareMathDelimiter{]}{\mathclose}{legacymaths}{93}{largesymbols}{3}
    \DeclareMathDelimiter{/}{\mathord}{legacymaths}{47}{largesymbols}{14}
     \DeclareMathSymbol{\mathdollar}{\mathord}{legacymaths}{36}
   \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
\SetSymbolFont{operators}{normal}\zf@enc\zf@rmmaths\mddefault\updefault
\SetMathAlphabet\mathrm{normal}\zf@enc\zf@rmmaths\mddefault\updefault
\SetMathAlphabet\mathit{normal}\zf@enc\zf@rmmaths\mddefault\itdefault
\SetMathAlphabet\mathbf{normal}\zf@enc\zf@rmmaths\bfdefault\updefault
\SetMathAlphabet\mathsf{normal}\zf@enc\zf@sfmaths\mddefault\updefault
\SetMathAlphabet\mathtt{normal}\zf@enc\zf@ttmaths\mddefault\updefault
\SetSymbolFont{operators}{bold}\zf@enc\zf@rmmaths\bfdefault\updefault
\ifdefined\zf@rmboldmaths
 \SetMathAlphabet\mathrm{bold}\zf@enc\zf@rmboldmaths\mddefault\updefault
```

```
    \SetMathAlphabet\mathbf{bold}\zf@enc\zf@rmboldmaths\bfdefault\updefault
    \SetMathAlphabet\mathit{bold}\zf@enc\zf@rmboldmaths\mddefault\itdefault
  \else
    \SetMathAlphabet\mathrm{bold}\zf@enc\zf@rmmaths\bfdefault\updefault
    \SetMathAlphabet\mathit{bold}\zf@enc\zf@rmmaths\bfdefault\itdefault
  \fi
  \SetMathAlphabet\mathsf{bold}\zf@enc\zf@sfmaths\bfdefault\updefault
  \SetMathAlphabet\mathtt{bold}\zf@enc\zf@ttmaths\bfdefault\updefault
  \let\font@warning\zf@font@warning}
```

`\zf@math@maybe` We're a little less sophisticated about not executing the `\zf@maths` macro if various other maths font packages are loaded. This list is based on the wonderful 'LaTeXFont Catalogue': `http://www.tug.dk/FontCatalogue/mathfonts.html`. I'm sure there are more I've missed. Do the TeX Gyre fonts have maths support yet?

Untested: would `\unless\ifnum\Gamma=28672\relax\@zf@mathfalse\fi` be a better test? This needs more cooperation with euler and lucida, I think.

```
\def\zf@math@maybe{%
  \@ifpackageloaded{anttor}{%
    \ifx\define@antt@mathversions a\@zf@mathfalse\fi}{}
  \@ifpackageloaded{arev}{\@zf@mathfalse}{}
  \@ifpackageloaded{eulervm}{\@zf@mathfalse}{}
  \@ifpackageloaded{mathdesign}{\@zf@mathfalse}{}
  \@ifpackageloaded{concmath}{\@zf@mathfalse}{}
  \@ifpackageloaded{cmbright}{\@zf@mathfalse}{}
  \@ifpackageloaded{mathesf}{\@zf@mathfalse}{}
  \@ifpackageloaded{gfsartemisia}{\@zf@mathfalse}{}
  \@ifpackageloaded{gfsneohellenic}{\@zf@mathfalse}{}
  \@ifpackageloaded{iwona}{%
    \ifx\define@iwona@mathversions a\@zf@mathfalse\fi}{}
  \@ifpackageloaded{kpfonts}{\@zf@mathfalse}{}
  \@ifpackageloaded{kmath}{\@zf@mathfalse}{}
  \@ifpackageloaded{kurier}{%
    \ifx\define@kurier@mathversions a\@zf@mathfalse\fi}{}
  \@ifpackageloaded{fouriernc}{\@zf@mathfalse}{}
  \@ifpackageloaded{fourier}{\@zf@mathfalse}{}
  \@ifpackageloaded{mathpazo}{\@zf@mathfalse}{}
  \@ifpackageloaded{mathptmx}{\@zf@mathfalse}{}
  \@ifpackageloaded{MinionPro}{\@zf@mathfalse}{}
  \@ifpackageloaded{unicode-math}{\@zf@mathfalse}{}
  \if@zf@math
    \zf@PackageInfo{%
      Adjusting the maths setup (use [no-math] to avoid this).}
    \zf@math
  \fi}
\AtBeginDocument{\zf@math@maybe}
```

8.10 Finishing up

Now we just want to set up loading the `.cfg` file, if it exists.

```
\if@zf@configfile
  \InputIfFileExists{fontspec.cfg}
    {\typeout{fontspec.cfg loaded.}}
    {\typeout{No fontspec.cfg file found; no configuration loaded.}}
\fi
```

The end! Thanks for coming.

File II

fontspec.cfg

As an example, and to avoid upsetting people as much as possible, I'm populating the default `fontspec.cfg` file with backwards compatibility feature aliases.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% FOR BACKWARDS COMPATIBILITY WITH PREVIOUS VERSIONS %%%

\let\newfontinstance\newfontfamily

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

\aliasfontfeature{BoldFont}{Bold}
\aliasfontfeature{ItalicFont}{Italic}
\aliasfontfeature{BoldItalicFont}{BoldItalic}
\aliasfontfeature{SmallCapsFont}{SmallCaps}
\aliasfontfeature{Style}{StyleOptions}
\aliasfontfeature{Contextuals}{Swashes}
\aliasfontfeatureoption{Contextuals}{Swash}{Contextual}
\aliasfontfeatureoption{Letters}{UppercaseSmallCaps}{SMALLCAPS}
\aliasfontfeatureoption{Letters}{UppercasePetiteCaps}{PETITECAPS}

%%%%%%%%%%%%%%%%%%%%%%%%
%%% FOR CONVENIENCE %%%

\newfontscript{Kana}{kana}
\newfontscript{Maths}{math}
\newfontscript{CJK}{hani}

```

File III

fontspec-example.ltx

```
\documentclass{article}

\usepackage{euler}
\usepackage[cm-default]{fontspec}
\usepackage{xltxtra}

\defaultfontfeatures{Scale=MatchLowercase,Mapping=tex-text}
\setmainfont[Numbers=Lowercase]{FPL Neu}
\setsansfont{Lucida Sans}
\setmonofont{Lucida Sans Typewriter}

\frenchspacing % TeX's default is a little old-fashioned...
```

```
13
14 \begin{document}
15 \pagestyle{empty}
16
17 \section*{The basics of the \textsf{fontspec} package}
18
19 The \textsf{fontspec} package enables automatic font selection
20 for \LaTeX{} documents typeset with \XeTeX{}. The basic command is\\
21 \indent \verb|\fontspec[font features]{font display name}|.\\
22 As an example:
23
24 \begin{center}
25   \Large
26   \fontspec[
27       Colour           = 0000CC,
28       Numbers          = OldStyle,
29       VerticalPosition = Ordinal,
30       Variant          = 2
31            ]{Apple Chancery}
32   My 1st example of Apple Chancery
33 \end{center}
34
35 The default, sans serif, and typewriter fonts may be set with the
36 \verb|\setmainfont|, \verb|\setsansfont| and \verb|\setmonofont|
37 commands, respectively, as shown in the preamble. They take the
38 same syntax as the \verb|\fontspec| package. All expected font
39 shapes are available:
40
41 \begin{center}
42   {\itshape Italics and \scshape small caps\dots}\\
43  {\sffamily\bfseries Bold sans serif and \itshape bold italic sans serif\dots}
44 \end{center}
45
46 With the roman and sans serif fonts set in the preamble, text fonts
47 in math mode are also changed: $\cos(n\pi)=\pm 1$. The maths
48 typeface `Euler' has been used in this document (with the \textsf{euler}
49 package---or the \textsf{eulervm} package if the |xpdfdvimx| driver
50 is being used), since the default Computer Modern maths font is rather light.
51 \[
52   \mathcal F(s) = \int^\infty_0 f(t) \exp(-st)\,\mathrm{d}t
53 \]
54
55 You'll also notice the \verb|\defaultfontfeatures| command in the preamble.
56 This command takes a single argument of font features that are then
57 applied to every subsequent instance of font selection. The first argument
58 in this case, \verb|Mapping=tex-text|, enables regular \TeX{} ligatures
59 like \verb|``---''| for ``---''. The second automatically scales the fonts
60 to the same x-height.
61
62 Please see the documentation for font feature explanation and further
63 package niceties.
```

```
64
65 \end{document}
```

Change History

v1.0

General: Initial version. 29

v1.1

General: Name change to fontspec. 29

`\setmainfont`: Implemented (with friends). 32

v1.10

General: `Color` brought back into the `.sty` 51

New feature `LetterSpace`. 50

Some `babel` encoding problems resolved. 31

`\addfontfeatures`: Saved family information macro changes. 33

`\zf@fontspec`: Saved family info split into two (now three) macros. 38

Space zapped from LaTeX family name due to various problems. 38

`\zf@make@feature`: Removed embarrassing space after warnings. 43

`\zf@math`: Added `lucimatx` checking. (Not really tested, though.) 61

Fixed Lucida bug (missing `\else`) 61

v1.11

General: `HyphenChar` checks its input now. 50

Added better support for Turkish language selection. 60

OpenType `Variant` fixed. 54

`\emph`: Redefinitions moved to `xltxtra`. 61

`\newfontface`: Name change from `\newfontfamily`. 33

`\newfontlanguage`: Fixed `\c@zf@language` setting not being global. 35

`\newfontscript`: Fixed `\c@zf@script` setting not being global. 35

`\zf@fontspec`: Ensure bold/italic fonts are loaded with the same renderer as the regular font even if unspecified. 38

`\zf@wordspace@parse`: Improved saving `\fontdimen` stuff to `\zd@adjust`(also see `PunctuationSpace`). 50

v1.12

General: `BoldFont`, etc., flags `\zf@nobf` conditional false rather than assuming it implicitly. This allows, *e.g.*, empty `BoldFont` to be overloaded. 47

Finally, use the `EU1` font encoding. 31

New feature `ExternalLocation` for loading external fonts. 46

Package option for disabling the EU1 encoding. 30

`\addfontfeatures`: Now use grouping to restore `\zf@default@option` change. 33

`\zf@make@aat@feature@string`: Fixed result of `\XeTeXfeaturename` output change (empty string if odd non-exclusive selector). 44

Removed `\@thisfontfeature` macro; replaced with `\@tempa`. 44

v1.13

General: `RawFeature` added 60

`SizeFeatures`: Beginnings of support for different font features for different font sizes, required by `unicode-math`. 48

`\zf@DeclareFontShape`: New feature `SizeFeatures` implemented. 40

`\zf@make@feature`: Warns (and fixes bug) when an AAT feature is requested for an ICU font and vice versa. 43

`\zf@partial@fontname`: Folded in the `\@tempa` `\let` command, changing the syntax. 48

v1.15

General: `RotatedGlyphs` now works for ICU fonts. 55

v1.16

`\zf@math@maybe`: Maths setup warning sent to the log file instead. 64