

The *fontspec* package

WILL ROBERTSON

2006/08/20 v1.12

Contents

1	Introduction	2	6.6	Contextuals	15
1.1	Usage	2	6.7	Vertical position	16
1.2	Warning	3	6.8	Fractions	17
1.3	About this manual	3	6.9	Variants	17
2	Brief overview	4	6.10	AAT Alternates	18
3	Font selection	4	6.11	Style	18
3.1	Default font families	4	6.12	Diacritics	19
3.2	Font instances for efficiency	5	6.13	Kerning	19
3.3	Arbitrary bold/italic/small caps fonts	5	6.14	CJK shape	20
3.4	Math(s) fonts	6	6.15	Character width	20
3.5	External fonts	6	6.16	Annotation	21
3.6	Miscellaneous font selecting details	7	6.17	Vertical typesetting	21
4	Selecting font features	7	6.18	AAT & Multiple Master font axes	22
4.1	Default settings	7	6.19	OpenType scripts and languages	22
4.2	Changing the currently selected features	8	7	Defining new features	24
4.3	Priority of feature selection	8	7.1	Renaming existing features & options	26
4.4	Different features for different font shapes	8	I	fontspec.sty	27
5	Font independent options	9	8	Implementation	27
5.1	Scale	9	8.1	Bits and pieces	27
5.2	Mapping	10	8.2	Option processing	28
5.3	Colour	10	8.3	Packages	28
5.4	Interword space	10	8.4	Encodings	28
5.5	Post-punctuation space	11	8.5	User commands	29
5.6	Letter spacing	11	8.6	Internal macros	33
5.7	The hyphenation character	11	8.7	keyval definitions	42
6	Font-dependent features	12	8.8	Italic small caps	55
6.1	Different font technologies: AAT and ICU	12	8.9	Selecting maths fonts	56
6.2	Optical font sizes	13	8.10	Finishing up	59
6.3	Ligatures	14	II	fontspec.cfg	61
6.4	Letters	14	III	fontspec-example.ltx	61
6.5	Numbers	15			

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{xltextra} % 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 xltextra 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.

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.
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 package² to be installed.

¹<http://scripts.sil.org/xetex>

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

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 T_EX font' without using the `xdvipdfmx` driver.

1.1.2 Configuration

If you wish to customise any part of the `fontspec` interface (see later in this manual, Section 7 on page 24 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 X_YT_EX. Either place it in the same folder as the main document for isolated cases, or in a location that X_YT_EX searches by default, *e.g.*, `~/Library/texmf/xelatex/`. The package option `[noconfig]` will suppress this behaviour under all circumstances.

1.1.3 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.

1.2 Warning

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

(→ v1.6: An example warning!)

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

1.3 About this manual

This document has been typeset with X_YT_EX using a variety of fonts to display various features that the package supports. You will not be able to typeset the documentation if you do not have all of these fonts, 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.

³An example is distributed with the package.

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}
\setromanfont[Mapping=tex-text]{Baskerville}
\setsansfont[Mapping=tex-text]{Skia}
\setmonofont{Courier}
```

3 Font selection

`\fontspec` `\fontspec[]{}` is the base command of the package, used for selecting the specified ** in a L^AT_EX family. The font features argument accepts comma separated *=<option>* lists; these will not be fully described until Section 6 on page 12.

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 X_YL^AT_EX in any way whatsoever: every font that is installed in the operating system may be accessed.

3.1 Default font families

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

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

```
\setromanfont{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 9, including methods for automatic scaling.

3.2 Font instances for efficiency

`\newfontfamily`
(→ v1.11: This macro used to be called `\newfontinstance`.
Backwards compatibility is preserved via `fontspec.cfg`.)

(←) 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:

This font is used for *notes*.

```
\newfontfamily\notefont{Didot}
\notefont This font is used for \emph{notes}.
```

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

`\newfontface`
(→ v1.11: New!)

(←) 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:

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

where is all the vegemite

This example is repeated in Section 6.6 on page 15.

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` features (←) are provided for these situations. If only one of these is used, the bold italic font is requested as the default from the *new* font.

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

```
\fontspec[BoldFont={Helvetica Neue}]
{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 <i>Italic</i> SemiBold <i>Italic</i>	<pre>\fontspec[BoldFont={* SemiBold}]{Baskerville} Baskerville \textit{Italic} \bfseries SemiBold \textit{Italic}</pre>
---	---

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:

Roman 123 SMALL CAPS 456	<pre>\fontspec[SmallCapsFont={Minion MM Small Caps & Oldstyle Figures},]{Minion MM Roman} Roman 123 \\\textsc{Small caps 456}</pre>
-----------------------------	---

3.4 Math(s) fonts

When `\setromanfont`, `\setsansfont` and `\setmonofont` are used in the preamble, they also define the fonts to be used in maths mode inside the `\mathrm`-type commands. This only occurs in the preamble because L^AT_EX 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.)

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

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

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

and this would allow you to typeset something like this:

$X \rightarrow X \rightarrow \mathbf{X}$ $\mathbf{X} \rightarrow \mathbf{X} \rightarrow \mathbf{X}$	<pre>\$ X \rightarrow \mathrm{X} \rightarrow \mathbf{X} \$ \\boldmath \$ X \rightarrow \mathrm{X} \rightarrow \mathbf{X} \$</pre>
--	---

3.5 External fonts

X_YL_AT_EX 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 \TeX , 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.

Optical sizes cannot be selected (either implicitly or explicitly) for externally loaded fonts; this is a failing of the `fontspec` package. Hopefully it will be rectified in the future.

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 12, 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.

<p>Some ‘default’ Didot 0123456789</p> <p>Now grey, with old-style figures: 0123456789</p>	<pre>\fontspec{Didot} Some ‘default’ Didot 0123456789 \defaultfontfeatures{Numbers=OldStyle, Colour=888888} \fontspec{Didot} Now grey, with old-style figures: 0123456789</pre>
--	---

4.2 Changing the currently selected features

`\addfontfeatures` The `\addfontfeatures{}` 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:

<p><i>In 1842, 999 people sailed 97 miles in 13 boats. In 1923, 111 people sailed 54 miles in 56 boats.’</i></p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>Year</th> <th>People</th> <th>Miles</th> <th>Boats</th> </tr> </thead> <tbody> <tr> <td>1842</td> <td>999</td> <td>75</td> <td>13</td> </tr> <tr> <td>1923</td> <td>111</td> <td>54</td> <td>56</td> </tr> </tbody> </table>	Year	People	Miles	Boats	1842	999	75	13	1923	111	54	56	<pre>\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}</pre>
Year	People	Miles	Boats										
1842	999	75	13										
1923	111	54	56										

`\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.

<p><i>ATTENTION ALL MARTINI DRINKERS</i></p> <p><i>ATTENTION ALL MARTINI DRINKERS</i></p>	<pre>\fontspec{Hoefler Text} \itshape \scshape Attention All Martini Drinkers \\\ \addfontfeature{ItalicFeatures={Alternate = 1}} Attention All Martini Drinkers \\\</pre>
---	--

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.

	<code>\fontspec[</code>
	<code>UprightFeatures={Colour = 220022,</code>
	<code>SmallCapsFeatures = {Colour=115511}},</code>
	<code>ItalicFeatures={Colour = 2244FF,</code>
	<code>SmallCapsFeatures = {Colour=112299}},</code>
	<code>BoldFeatures={Colour = FF4422,</code>
	<code>SmallCapsFeatures = {Colour=992211}},</code>
	<code>BoldItalicFeatures={Colour = 888844,</code>
	<code>SmallCapsFeatures = {Colour=444422}},</code>
	<code>]{Hoefer Text}</code>
Upright SMALL CAPS	<code>Upright {\scshape Small Caps}\\\</code>
Italic ITALIC SMALL CAPS	<code>\itshape Italic {\scshape Italic Small Caps}\\\</code>
Bold BOLD SMALL CAPS	<code>\upshape\bfseries Bold {\scshape Bold Small Caps}\\\</code>
Bold Italic BOLD ITALIC SMALL CAPS	<code>\itshape Bold Italic {\scshape Bold Italic Small Caps}</code>

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 X_YT_EX, 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 now (←) also takes the options `MatchLowercase` and `MatchUppercase`, 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.

	<code>\setromanfont{Georgia}</code>
	<code>\newfontfamily\lc[Scale=MatchLowercase]{Verdana}</code>
The perfect match is hard to find.	<code>The perfect match {\lc is hard to find.}\\\</code>
LOGO FONT	<code>\newfontfamily\uc[Scale=MatchUppercase]{Arial}</code>
	<code>L O G O \uc F O N T</code>

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 X_YTeX 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 7 and Section 6 on page 12, uses X_YTeX 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 T_EX 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 X_YTeX, 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 X_YTeX 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 T_EX, 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.

That 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 14).

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 character⁴ 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.

<p>A MULTITUDE OF OBSTREPEROUSLY HYPHENATED ENTITIES</p> <p>A MULTITUDE OF OBSTREPER- OUSLY HYPHENATED ENTITIES</p> <p>A MULTITUDE OF OBSTREPER- OUSLY HYPHENATED ENTITIES</p>	<pre>\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</pre>
--	---

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 14).

The xltextra package redefines L^AT_EX’s \- macro such that it adjusts along with the above changes.

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:

*My 1st example of
Apple Chancery*

```
\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 X_YL^AT_EX 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 1.1.3 on page 3 these warnings can be suppressed by selecting the [quiet] package option.

6.1 Different font technologies: AAT and ICU

X_YL^AT_EX 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.

⁴I found the character, and its number, in Mac OS X’s Character Palette.

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 22 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 X₃TeX *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:

	<code>\fontspec[OpticalSize=0]{Warnock Pro Caption}</code>	
Warnock Pro optical sizes	Warnock Pro optical sizes	<code>\\</code>
Warnock Pro optical sizes	<code>\fontspec[OpticalSize=0]{Warnock Pro}</code>	
Warnock Pro optical sizes	Warnock Pro optical sizes	<code>\\</code>
Warnock Pro optical sizes	<code>\fontspec[OpticalSize=0]{Warnock Pro Subhead}</code>	
Warnock Pro optical sizes	Warnock Pro optical sizes	<code>\\</code>
	<code>\fontspec[OpticalSize=0]{Warnock Pro Display}</code>	
	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)

	<code>\fontspec{Warnock Pro}</code>	
Automatic optical size	Automatic optical size	<code>\\</code>
Automatic optical size	<code>\scalebox{0.4}{\Huge</code>	
	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 22 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 L^AT_EX, and the optical size for a Multiple Master font must always be specified explicitly.

MM optical size test	<code>\fontspec[OpticalSize=11]{Minion MM Roman}</code>
MM optical size test	MM optical size test
MM optical size test	<code>\fontspec[OpticalSize=47]{Minion MM Roman}</code>
	MM optical size test
	<code>\fontspec[OpticalSize=71]{Minion MM Roman}</code>
	MM optical size test


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

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

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

Some other Apple AAT fonts have those ‘Rare’ ligatures contained in the Icelandic feature. Notice also that the old T_EX trick of splitting up a ligature with an empty brace pair does not work in X_YT_EX; you must use a `opt 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 feature (←) specifies how the letters in the current font will look. For AAT fonts, you may choose from Normal, Uppercase, Lowercase, SmallCaps, and InitialCaps.

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

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


(→ 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 Unicaise. (←) 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’). ‘Unicaise’ is a weird hybrid of upper and lower case letters.

THIS SENTENCE NO VERB
THIS SENTENCE no verb

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

The Uppercase option is also provided *but* it will (probably) not actually map letters to uppercase.⁵ 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 19). 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!)

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 0123456789

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

⁵If you want automatic uppercase letters, look to L^AT_EX’s \MakeUppercase command.

where is all the vegemite

```
\newfontface\fancy
[Contextuals={WordInitial,WordFinal}]
{Hoefler Text Italic}
\fontspec\newfontface\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.
```

(→ v1.9: Used to be Contextual;
still works.)

For OpenType fonts, all features as above but the LineInitial feature are supported, and Swash turns on contextual swashes (←).

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.

6.7 Vertical position

The VerticalPosition feature is used to access things like subscript (Superior) and superscript (Inferior) 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!

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

6.8 Fractions

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

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

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

$\frac{1}{2}$ $\frac{5}{6}$
1/2 5/6

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

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

$\frac{13579}{24680}$
 $\frac{13579}{24680}$

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

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

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

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

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

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

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

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

$\frac{13579}{24680}$

```
\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 24 for a way to assign names to variants, which should be done on a per-font basis.

6.10 AAT Alternates

Selection of Alternates in AAT fonts *again* must be done numerically.

<i>Sphinx Of Black Quartz, JUDGE My Vow</i>	<code>\fontspec[Alternate=0]{Hoefer Text Italic}</code>
<i>Sphinx Of Black Quartz, JUDGE My Vow</i>	<code>Sphinx Of Black Quartz, {\scshape Judge My Vow} \</code>
<i>Vow</i>	<code>\fontspec[Alternate=1]{Hoefer Text Italic}</code>
	<code>Sphinx Of Black Quartz, {\scshape Judge My Vow}</code>

See Section 7 on page 24 for a way to assign names to alternates, which should be done on a per-font basis.

6.11 Style

(→ v1.7: The old name, `StyleOptions`, still works.) The options of the Style feature (←) are defined in AAT as one of the following: Display, Engraved, IlluminatedCaps, Italic, Ruby,⁶ TallCaps, or TitlingCaps.

<i>{ABCD...WXYZ}</i>	<code>\newfontface\officedoor[Style=Engraved]{Hoefer Text}</code>
	<code>\officedoor [ABCD\dots WXYZ]</code>

ICU supported options are Alternate, Italic, Historic, Ruby,⁶ Swash, TitlingCaps, HorizontalKana, and VerticalKana.

<i>KQRkvw y</i>	<code>\fontspec{Warnock Pro}</code>
<i>KQRkvw y</i>	<code>K Q R k v w y \</code>
	<code>\addfontfeature{Style=Alternate}</code>
	<code>K Q R k v w y</code>

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

<i>MQZ</i>	<code>\fontspec{Adobe Jenson Pro}</code>
<i>MQZ</i>	<code>M Q Z \</code>
	<code>\addfontfeature{Style=Historic}</code>
	<code>M Q Z</code>

<i>TITLING CAPS</i>	<code>\fontspec{Adobe Garamond Pro}</code>
<i>TITLING CAPS</i>	<code>TITLING CAPS \</code>
	<code>\addfontfeature{Style=TitlingCaps}</code>
	<code>TITLING CAPS</code>

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

<i>Latin ようこそ ワカヨタレソ</i>	<code>\fontspec{Hiragino Mincho Pro W3}</code>
<i>Latin ようこそ ワカヨタレソ</i>	<code>Latin ようこそ ワカヨタレソ \</code>
	<code>\addfontfeature{Style={Italic, Ruby}}</code>
	<code>Latin ようこそ ワカヨタレソ</code>

⁶‘Ruby’ refers to a small optical size, used in Japanese typography for annotations.

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 T_EX input conventions for obtaining such characters instead.

Ó Ö Ø
 O' O" O/
 Better: Ó Ö Ø

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

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

No options for OpenType fonts.

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 14, 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 glyphs 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.)

唾嚙軀 妍并訝
唾嚙軀 妍并訝
啞嚙軀 妍并訝

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

6.15 Character width

Many Asian fonts are equipped with variously spaced characters for shoe-horning into their generally monospaced text. These are accessed through the CharacterWidth feature.⁷ (←) 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.)

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.

			\def\test{\makebox[2cm][l]{ようこそ}%
			\makebox[2.5cm][l]{ワカヨタレソ}%
		abcdef	\makebox[2.5cm][l]{abcdef}%
ようこそ	ワカヨタレソ	abcdef	\fontspec{Hiragino Mincho Pro}
ようこそ	ワカヨタレソ	a b c d e f	{\addfontfeature{CharacterWidth=Proportional}\test}\
ようこそ	ワカヨタレソ	abcdef	{\addfontfeature{CharacterWidth=Full}\test}\
			{\addfontfeature{CharacterWidth=Half}\test}

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

⁷Apple 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.

— 1 2 3 2 1 —
 -1234554321-
 -123456787654321-
 -12345678900987654321-

```
\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

A recent feature of \XeTeX is the ability to rotate the glyphs in AAT fonts by 90° , providing a method to typeset vertically by building a horizontal box as normal and then rotating it.

共產主義者は

共
産
主
義
者
は

```
\fontspec{Hiragino Mincho Pro}
共產主義者は

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

The AAT renderer is required above because Xe_{La}TeX chooses the ICU renderer by preference when both options are available; if it is not explicitly chosen, the glyphs will not be rotated and a warning will be printed in the output.

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 Xe_{La}TeX 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	<code>\fontspec[Weight=0.5,Width=3]{Skia}</code>	
Really fat and condensed Skia	<code>\fontspec[Weight=2,Width=0.5]{Skia}</code>	<code>\\</code>

Variations along a multiple master font's optical size axis has been shown previously in Section 6.2 on page 13.

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 1 on page 24 and Table 2 on page 25, 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 12.

6.19.1 Script examples

In the following examples, the same font is used to typeset the verbatim input and the Xe_{La}TeX 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	<code>\fontspec[Script=Arabic]{Code2000}</code>	العربي
हिन्दी	<code>\fontspec[Script=Devanagari]{Code2000}</code>	हिन्दी

লেখ	<code>\fontspec[Script=Bengali]{Code2000}</code> লেখ
મય્યાદા-સૂચક નિવેદન	<code>\fontspec[Script=Gujarati]{Code2000}</code> મય્યાદા-સૂચક નિવેદન
നമുടെ പാരമ്പര്യ	<code>\fontspec[Script=Malayalam]{Code2000}</code> നമുടെ പാരമ്പര്യ
ਆਦਿ ਸਚੁ ਜੁਗਾਦਿ ਸਚੁ	<code>\fontspec[Script=Gurmukhi]{Code2000}</code> ਆਦਿ ਸਚੁ ਜੁਗਾਦਿ ਸਚੁ
தமிழ் தேடி	<code>\fontspec[Script=Tamil]{Code2000}</code> தமிழ் தேடி
תורה	<code>\fontspec[Script=Hebrew]{Code2000}</code> תורה

6.19.2 Language examples

Vietnamese:

cáp sỏ mỗỉ	<code>\fontspec{Doulos SIL}</code>
cáp sỏ mỗỉ	<code>cáp sỏ mỗỉ \</code>
	<code>\addfontfeature{Language=Vietnamese}</code>
	<code>cáp sỏ mỗỉ</code>

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

Ş ş Ţ ţ	<code>\fontspec{FPL Neu}</code>
Ş ş Ţ ţ	<code>Ş ş Ţ ţ \</code>
	<code>\addfontfeature{Language=Moldavian}</code>
	<code>Ş ş Ţ ţ</code>

6.19.3 Defining new scripts and languages

`\newfontscript` 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 on the next page) is the error-checking that is performed when the script or language is requested.

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 X_YTeX file AAT-info.tex to obtain the code numbers. For example:

This is XeTeX by Jonathan Kew. `\newAATfeature{Alternate}{HoeflerSwash}{17}{1}`
`\fontspec[Alternate=HoeflerSwash]{Hoefler Text Italic}`
 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 22) must be activated to achieve the effect shown.

`\newICUfeature{Style}{NoLocalForms}{-loc1}`
`\fontspec[Language=Moldavian]{FPL Neu}`
`$ $ T t` `\`
`\addfontfeature{Style=NoLocalForms}`
`$ $ T t`

`\newfontfeature`

In case the above commands do not accommodate the desired font feature (perhaps a new X_YTeX 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:

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

Table 1: Defined Scripts for OpenType fonts. Aliased names are shown in adjacent positions marked with red pilcrow (¶), defined in `fontspec.cfg`.

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

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

sockdolager rubdown
 sockdolager rubdown

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

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{}{<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}
```

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.)

8.1 Bits and pieces

Conditionals

```
1 \newif\ifzf@firsttime
2 \newif\ifzf@nobf
3 \newif\ifzf@noit
4 \newif\ifzf@nosc
5 \newif\ifzf@tfm
6 \newif\ifzf@atsui
7 \newif\ifzf@icu
8 \newif\ifzf@mm
```

For dealing with legacy maths

```
9 \newif\ifzf@math@euler
10 \newif\ifzf@math@lucida
11 \newif\ifzf@package@euler@loaded
```

For, well, dealing with babel:

```
12 \newif\ifzf@package@babel@loaded
```

For package options:

```
13 \newif\if@zf@configfile
14 \newif\if@zf@euenc
```

Counters

```
15 \newcount\c@zf@newff
16 \newcount\c@zf@index
17 \newcount\c@zf@script
18 \newcount\c@zf@language
```

fontspec shorthands:

```
19 \newcommand\zf@PackageError[2]{\PackageError{fontspec}{#1}{#2}}
20 \newcommand\zf@PackageWarning[1]{\PackageWarning{fontspec}{#1}}
21 \newcommand\zf@PackageInfo[1]{\PackageInfo{fontspec}{#1}}
```

```
\def@cx LATEX3-like syntax for various low level commands. Makes life much easier; can't
\gdef@cx wait for the official interface :)
\let@cc 22 \providecommand\def@cx[2]{\expandafter\edef\csname#1\endcsname{#2}}
```

```

23 \providecommand\gdef@cx[2]{\expandafter\xdef\csname#1\endcsname{#2}}
24 \providecommand\let@cc[2]{%
25   \expandafter\let\csname#1\expandafter\endcsname\csname#2\endcsname}

```

8.2 Option processing

```

26 \DeclareOption{cm-default}{\@zf@euencfalse}
27 \DeclareOption{lm-default}{\@zf@euenctrue}
28 \DeclareOption{config}{\@zf@configfiletrue}
29 \DeclareOption{noconfig}{\@zf@configfilefalse}
30 \DeclareOption{quiet}{\let\zf@PackageWarning\zf@PackageInfo}
31 \ExecuteOptions{config,lm-default}
32 \ProcessOptions*

```

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

```

33 \RequirePackage{ifxetex}
34 \RequireXeTeX

```

8.3 Packages

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

```

35 \RequirePackage{calc}
36 \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 X_YTeX 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.

```

37 \if@zf@euenc
38   \def\zf@enc{EU1}
39   \RequirePackage{lmodern}
40   \RequirePackage[\zf@enc]{fontenc}
41 \else
42   \def\zf@enc{U}
43   \let\encodingdefault\zf@enc
44 \fi
45 \let\UTFencname\zf@enc

```

Dealing with a couple of the problems introduced by `babel`:

```

46 \let\cyrillicencoding\zf@enc
47 \let\latinencoding\zf@enc
48 \g@addto@macro\document{%
49   \let\cyrillicencoding\zf@enc
50   \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 33.

8.5.1 Font selection

<code>\fontspec</code>	<p>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 <code>\zf@fontspec</code>, which takes the same arguments as the top level macro and puts the new-fangled font family name into the global <code>\zf@family</code>. Then this new font family is selected.</p> <pre> 51 \newcommand*\fontspec[2] [] {% 52 \zf@fontspec{#1}{#2}% 53 \fontfamily\zf@family\selectfont 54 \ignorespaces} </pre>
<code>\setromanfont</code> <code>\setsansfont</code> <code>\setmonofont</code>	<p>The following three macros perform equivalent operations setting the default font (using <code>\let</code> rather than <code>\renewcommand</code> because <code>\zf@family</code> will change in the future) for a particular family: roman, sans serif, or typewriter (monospaced). I end them with <code>\normalfont</code> so that if they’re used in the document, the change registers immediately.</p> <pre> 55 \newcommand*\setromanfont[2] [] {% 56 \zf@fontspec{#1}{#2}% 57 \let\rmdefault\zf@family 58 \normalfont} 59 \newcommand*\setsansfont[2] [] {% 60 \zf@fontspec{#1}{#2}% 61 \let\sfdefault\zf@family 62 \normalfont} 63 \newcommand*\setmonofont[2] [] {% 64 \zf@fontspec{#1}{#2}% 65 \let\ttdefault\zf@family 66 \normalfont} </pre>
<code>\setmathrm</code> <code>\setmathsf</code> <code>\setboldmathrm</code> <code>\setmathtt</code>	<p>These commands are analogous to <code>\setromanfont</code> and others, but for selecting the font used for <code>\mathrm</code>, <i>etc.</i> They can only be used in the preamble of the document. <code>\setboldmathrm</code> is used for specifying which fonts should be used in <code>\boldmath</code>.</p> <pre> 67 \newcommand*\setmathrm[2] [] {% 68 \zf@fontspec{#1}{#2}% 69 \let\zf@rmmaths\zf@family} 70 \newcommand*\setboldmathrm[2] [] {% 71 \zf@fontspec{#1}{#2}% 72 \let\zf@rmboldmaths\zf@family} 73 \newcommand*\setmathsf[2] [] {% 74 \zf@fontspec{#1}{#2}% 75 \let\zf@sfmaths\zf@family} </pre>

```

76 \newcommand*\setmathtt[2]{}{%
77   \zf@fontspec{#1}{#2}%
78   \let\zf@ttmaths\zf@family}
79 \@onlypreamble\setmathrm
80 \@onlypreamble\setboldmathrm
81 \@onlypreamble\setmathsf
82 \@onlypreamble\setmathtt

```

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

```

83 \def\zf@rmmaths{\rmdefault}
84 \def\zf@sffmaths{\sfdefault}
85 \def\zf@ttmaths{\ttdefault}

```

`\newfontfamily` This macro takes the arguments of `\fontspec` with a prepended *<instance cmd>* (code for middle optional argument generated by Scott Pakin’s `newcommand.py`). `\newfontface` 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.

```

86 \newcommand*\newfontfamily[1]{%
87   \@ifnextchar[{\newfontfamily@i#1}{\newfontfamily@i#1[]}}
88 \def\newfontfamily@i#1[#2]#3{%
89   \zf@fontspec{#2}{#3}%
90   \edef\@tempa{%
91     \noexpand\DeclareRobustCommand\noexpand#1
92     {\noexpand\fontfamily{\zf@family}\noexpand\selectfont}}%
93   \@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.

```

94 \newcommand*\newfontface[1]{%
95   \@ifnextchar[{\newfontface@i#1}{\newfontface@i#1[]}}
96 \def\newfontface@i#1[#2]#3{%
97   \zf@fontspec{BoldFont={},ItalicFont={},SmallCapsFont={},#2}{#3}%
98   \edef\@tempa{%
99     \noexpand\DeclareRobustCommand\noexpand#1
100     {\noexpand\fontfamily{\zf@family}\noexpand\selectfont}}%
101   \@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.

```

102 \newcommand*\defaultfontfeatures[1]{\def\zf@default@options{#1,}}
103 \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.

```

104 \newcommand*\addfontfeatures[1]{%
105   \begingroup
106     \let\zf@default@options\@empty
107     \edef\@tempa{%
108       \noexpand\zf@fontspec
109       {\csname zf@family@options\fontfamily\endcsname,#1}%
110       {\csname zf@family@fontname\fontfamily\endcsname}}%
111     \@tempa
112   \endgroup
113   \fontfamily\zf@family\selectfont
114   \ignorespaces}
115 \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.

```

116 \newcommand*\newfontfeature[2]{%
117   \stepcounter{zf@newff}%
118   \def@cx{+zf-#1}{+zf-\the\c@zf@newff}%
119   \define@key{zf}[options]{#1}[]{}%
120   \zf@update@family{\csname+zf-#1\endcsname}%
121   \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.

```

122 \newcommand*\newAATfeature[4]{%
123   \unless\ifcsname zf@options@#1\endcsname
124     \zf@define@font@feature{#1}%
125     \fi
126   \key@ifundefined{zf}[#1]{#2}{#3}{%

```



```

127 \zf@PackageWarning{Option '#2' of font feature '#1' overwritten.}}%
128 \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.

```

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

```

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

```

136 \newcommand*\aliasfontfeature[2]{\multi@alias@key{#1}{#2}}
137 \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 (X_YTEX) `\font` feature string with the appropriate script selection tag.

```

138 \newcommand*\newfontscript[2]{%
139 \define@key{zf@feat}{Script}{#1}[]{}%
140 \zf@check@ot@script{#2}%
141 \if@tempswa
142 \global\c@zf@script\@tempcnta\relax
143 \xdef\zf@script@name{#1}%
144 \xdef\zf@family@long{\zf@family@long+script=#1}%
145 \xdef\zf@pre@ff{script=#2,\zf@pre@ff}%
146 \else
147 \zf@PackageWarning{Font \fontname\zf@basefont does not con-
tain script '#1'}%
148 \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 (X_YTEX) `\font` feature string with the appropriate language selection tag.

```

149 \newcommand*\newfontlanguage[2]{%
150 \define@key{zf@feat}{Lang}{#1}[]{}%
151 \zf@check@ot@lang{#2}%
152 \if@tempswa
153 \global\c@zf@language\@tempcnta\relax
154 \xdef\zf@language@name{#1}%
155 \xdef\zf@family@long{\zf@family@long+lang=#1}%

```

```

156     \xdef\zf@pre@ff{\zf@pre@ff language=#2,}%
157   \else
158     \zf@PackageWarning{Font \fontname\zf@basefont does not contain
159                       language '#1' for script '\zf@script@name'}%
160   \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.

```

161 \newcommand*\zf@fontspec[2]{%
162   \begingroup
163   \zf@init
164   \edef\zf@fontname{#2}%
165   \let\zf@family@long\zf@fontname
166   \setkeys*[zf]{preparse}{#1}%
167   \let\zf@font@feat\XKV@rm

```

After pre-parsing, check if an external font is to be loaded. This only works with the `xdvi`pdfmx driver, by the way.

```

168   \unless\ifx\zf@font@path\relax
169     \def\zf@font@wrap##1{[\zf@font@path##1]}%
170   \fi

```

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.

```

171   \font\zf@basefont="\zf@font@wrap\zf@fontname\zf@suffix" at \f@size pt
172   \zf@set@font@type
173   \ifx\zf@suffix\empty
174     \ifzf@atsui
175       \def\zf@suffix{/AAT}%
176   \else
177     \ifzf@icu
178       \def\zf@suffix{/ICU}%

```

```

179     \fi
180   \fi
181   \font\zf@basefont="\zf@font@wrap\zf@fontname\zf@suffix" at \f@size pt
182 \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 `makefamilyxkeyval\setkeys` declaration. Probably.

```

183 \zf@firsttimetrue
184   \zf@get@feature@requests{\zf@font@feat}%
185 \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.

```

186 \unless\ifcsname zf@UID@\zf@family@long\endcsname
187   \ifcsname c@zf@famc@#2\endcsname
188     \expandafter\stepcounter\else
189     \expandafter\newcounter\fi
190     {zf@famc@#2}%
191   \def@cx{zf@UID@\zf@family@long}{%
192     \zap@space#2 \@empty
193     (\expandafter\the\csname c@zf@famc@#2\endcsname)}%
194 \fi
195 \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`.

```

196 \unless\ifcsname zf@family@fontname\zf@family\endcsname
197   \zf@PackageInfo{Defining font family for "#2"
198     with options [\zf@default@options #1]}%
199   \gdef@cx{zf@family@fontname\zf@family}{\zf@fontname}%
200   \gdef@cx{zf@family@options\zf@family}{\zf@default@options #1}%
201   \gdef@cx{zf@family@fontdef\zf@family}
202     {\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`.)

```
203 \DeclareFontFamily{\zf@enc}{\zf@family}{}%
204 \zf@make@font@shapes{\zf@fontname}
205 {\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.

```
206 \unless\ifzf@nobf
207 \ifx\zf@bf\@empty
208 \zf@make@font@shapes[\zf@fontname]{/B}
209 {\bfdefault}{\updefault}{\zf@font@feat\zf@bf@feat}%
210 \else
211 \zf@make@font@shapes{\zf@bf}
212 {\bfdefault}{\updefault}{\zf@font@feat\zf@bf@feat}%
213 \fi
214 \fi
```

And italic in the same way:

```
215 \unless\ifzf@noit
216 \ifx\zf@it\@empty
217 \zf@make@font@shapes[\zf@fontname]{/I}
218 {\mddefault}{\itdefault}{\zf@font@feat\zf@it@feat}%
219 \else
220 \zf@make@font@shapes{\zf@it}
221 {\mddefault}{\itdefault}{\zf@font@feat\zf@it@feat}%
222 \fi
223 \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 occurrence, presumably), the new bold font is used to define the new bold italic font.

```
224 \@tempswatrue
225 \ifzf@nobf\@tempswafalse\fi
226 \ifzf@noit\@tempswafalse\fi
227 \if@tempswa
228 \ifx\zf@bfit\@empty
229 \ifx\zf@bf\@empty
230 \ifx\zf@it\@empty
231 \zf@make@font@shapes[\zf@fontname]{/BI}
232 {\bfdefault}{\itdefault}{\zf@font@feat\zf@bfit@feat}%
233 \else
234 \zf@make@font@shapes[\zf@it]{/B}
235 {\bfdefault}{\itdefault}{\zf@font@feat\zf@bfit@feat}%
236 \fi
237 \else
238 \zf@make@font@shapes[\zf@bf]{/I}
```

```

239         {\bfdefault}{\itdefault}{\zf@font@feat\zf@bfit@feat}%
240         \fi
241     \else
242         \zf@make@font@shapes{\zf@bfit}
243         {\bfdefault}{\itdefault}{\zf@font@feat\zf@bfit@feat}%
244     \fi
245 \fi
246 \fi
247 \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.

```

248 \newcommand*\zf@set@font@type{%
249     \zf@tfmfalse \zf@atsuifalse \zf@icufalse \zf@mmfalse
250     \ifcase\XeTeXfonttype\zf@basefont
251         \zf@tfm
252     \or
253         \zf@atsuitrue
254         \ifnum\XeTeXcountvariations\zf@basefont > 0
255             \zf@mmtrue
256         \fi
257     \or
258         \zf@icutrue
259     \fi}

```

`\zf@make@font@shapes` This macro uses `\DeclareFontShape` to define the font shape in question. The arguments are:

- #1#2 the font name,
- #3 the font series,
- #4 the font shape, and
- #5 the font features.

The optional first argument is used when making the font shapes for bold, italic, and bold italic fonts using X_YTEX'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.

```

260 \newcommand*\zf@make@font@shapes[5][{}]{%
261     \begingroup
262     \edef\@tempa{#1}%
263     \unless\ifx\@tempa\@empty
264         \font\@tempfonta="\zf@font@wrap{#1}\zf@suffix" at \f@size pt
265         \edef\@tempa{\fontname\@tempfonta}%
266     \fi
267     \font\@tempfontb="\zf@font@wrap{#1#2}\zf@suffix" at \f@size pt
268     \edef\@tempb{\fontname\@tempfontb}%
269     \ifx\@tempa\@tempb

```

```

270 \zf@PackageInfo{Could not resolve font #1#2 (it might not exist)}%
271 \else
272 \edef\zf@fontname{#1#2}%
273 \let\zf@basefont\@tempfontb
274 \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 55 for the code that enables this usage.

```

275 \ifx\zf@sc\@empty
276 \unless\ifzf@nosc
277 \zf@make@smallcaps
278 \ifx\zf@smallcaps\@empty\else
279 \zf@DeclareFontShape[\zf@smallcaps]{#3}
280 {\ifx#4\itdefault\sidefault\else\scdefault\fi}{#5\zf@sc@feat}%
281 \fi
282 \fi
283 \else
284 \edef\zf@fontname{\zf@sc}%
285 \zf@DeclareFontShape{#3}
286 {\ifx#4\itdefault\sidefault\else\scdefault\fi}{#5\zf@sc@feat}%
287 \fi
288 \fi
289 \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` Wrapper for `\DeclareFontShape`. Among omitting common arguments, it also fully expands its input upon execution, which is required to save the contents of `\zf@adjust` at the time of processing to the font definition.

The 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.

```

290 \newcommand\zf@DeclareFontShape[4][]{%
291 \zf@get@feature@requests{#4}%
292 \def\@tempb{"\zf@font@wrap\zf@fontname\zf@suffix:\zf@pre@ff\zf@ff#1"}%
293 \zf@PackageInfo{\string\font\space is \@tempb}%
294 \edef\@tempa{\noexpand
295 \DeclareFontShape{\zf@enc}{\zf@family}{#2}{#3}
296 {\<->\zf@scale\@tempb}{\zf@adjust}}%
297 \@tempa
298 \edef\@tempa{#3}\edef\@tempb{\itdefault}%
299 \ifx\@tempa\@tempb
300 \edef\@tempa{\noexpand
301 \DeclareFontShape{\zf@enc}{\zf@family}{#2}{\sldefault}
302 {\<->sub*\zf@family/#2/\itdefault}{\zf@adjust}}%
303 \@tempa
304 \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.

```

305 \newcommand*\zf@update@family}[1]{%
306   \ifzf@firsttime
307     \xdef\zf@family@long{\zf@family@long#1}%
308   \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.

```

309 \newcommand*\zf@get@feature@requests[1]{%
310   \let\zf@ff      \@empty
311   \let\zf@scale   \@empty
312   \let\zf@adjust  \@empty
313   \edef\@tempa{\noexpand\setkeys[zf]{options}{\zf@default@options#1}}%
314   \@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.

```

315 \newcommand*\zf@init{%
316   \let\zf@pre@ff      \@empty
317   \let\zf@font@feat   \@empty
318   \let\zf@font@path   \@empty
319   \let\zf@font@wrap   \@firstofone
320   \let\zf@suffix      \@empty
321   \let\zf@bf          \@empty
322   \let\zf@it          \@empty
323   \let\zf@bfit        \@empty
324   \let\zf@sc          \@empty
325   \let\zf@up@feat     \@empty
326   \let\zf@bf@feat     \@empty
327   \let\zf@it@feat     \@empty
328   \let\zf@bfit@feat   \@empty
329   \let\zf@sc@feat     \@empty
330   \c@zf@script 1818326126\relax
331   \def\zf@script@name{Latin}%
332   \c@zf@language 0\relax
333   \def\zf@language@name{Default}%
334 }

```

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

```

335 \newcommand*\zf@make@smallcaps{%
336   \let\zf@smallcaps\@empty
337   \ifzf@atsui
338     \zf@make@aat@feature@string{3}{3}%
339     \unless\ifx\@tempa\@empty

```

```

340     \edef\zf@smallcaps{\@tempa;}%
341     \fi
342 \fi
343 \ifzf@icu
344     \zf@check@ot@feat{+smcp}%
345     \if@tempswa
346         \edef\zf@smallcaps{+smcp,}%
347     \fi
348 \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.

```

349 \newcommand*\zf@update@ff[1]{%
350     \unless\ifzf@firsttime
351         \xdef\zf@ff{\zf@ff #1\ifzf@icu,\else;\fi}%
352     \fi}

```

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

```

353 \newcommand*\zf@make@feature[3]{%
354     \ifzf@atsui
355         \zf@make@aat@feature@string{#1}{#2}%
356         \ifx\@tempa\@empty
357             \zf@PackageWarning{%
358                 AAT feature '\XKV@tfam=\XKV@tkey'
359                 (#1,#2) not available in font \fontname\zf@basefont}%
360         \else
361             \zf@update@family{+#1,#2}%
362             \zf@update@ff\@tempa
363         \fi
364     \fi
365     \ifzf@icu
366         \zf@check@ot@feat{#3}%
367         \if@tempswa
368             \zf@update@family{#3}%
369             \zf@update@ff{#3}%
370         \else
371             \zf@PackageWarning{%
372                 OpenType feature '\XKV@tfam=\XKV@tkey' (#3)
373                 not available in font \fontname\zf@basefont, script
374                 '\zf@script@name', language '\zf@language@name'}%
375         \fi
376     \fi}

```

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

```

\zf@define@feature@option 377 \newcommand*\zf@define@font@feature[1]{%
378     \define@key[zf]{options}{#1}{\setkeys[zf@feat]{#1}{##1}}}
379 \newcommand*\zf@define@feature@option[5]{%
380     \define@key[zf@feat]{#1}{#2}[]{\zf@make@feature{#3}{#4}{#5}}}

```


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

```
381 \newcommand*\keyval@alias@key[4][KV]{%
382   \let@cc{#1#2#4}{#1#2#3}%
383   \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.

```
384 \newcommand*\multi@alias@key[2]{
385   \key@ifundefined[zf]{preparse}{#1}
386   {\key@ifundefined[zf]{options}{#1}
387    {\zf@PackageError{The feature #1 doesn't appear to be defined}
388     {It looks like you're trying to rename a feature that doesn't exist.}}
389    {\keyval@alias@key[zf]{options}{#1}{#2}}}
390   {\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 38).

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

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

```
394   \ifnum\XeTeXisexclusivefeature\zf@basefont #1 > 0
395     \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:

```
396   \else
397     \unless\ifodd #2
398       \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.

```
399   \else
400     \edef\@tempb{\XeTeXselectorname\zf@basefont #1 \numexpr#2-1\relax}%
401     \unless\ifx\@tempb\@empty
402       \edef\@tempb{!\@tempb}%
403   \fi
404   \fi
```

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

```
405   \fi
406   \unless\ifx\@tempb\@empty
407     \edef\@tempa{\@tempa=\@tempb}%
408   \else
409     \let\@tempa\@empty
```

```

410 \fi
411 \fi}

```

`\zf@iv@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.

```

412 \newcommand\zf@iv@strnum[1]{%
413   \zf@iv@strnum@i#1 \@nil}
414 \def\zf@iv@strnum@i#1 \@nil{%
415   \zf@iv@strnum@ii#1\@empty\@empty\@nil}
416 \def\zf@iv@strnum@ii#1#2#3#4#5\@nil{%
417   \@tempcnta\z@
418   \@tempcntb`#1\relax
419   \multiply\@tempcntb"1000000\advance\@tempcnta\@tempcntb
420   \@tempcntb`#2
421   \multiply\@tempcntb"10000\advance\@tempcnta\@tempcntb
422   \expandafter\@tempcntb\ifx\@empty#332\else`#3\fi
423   \multiply\@tempcntb"100\advance\@tempcnta\@tempcntb
424   \expandafter\@tempcntb\ifx\@empty#432\else`#4\fi
425   \advance\@tempcnta\@tempcntb}
426 \newcommand\zf@v@strnum[1]{%
427   \expandafter\zf@iv@strnum@i\@gobble#1 \@nil}

```

`\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.

```

428 \newcommand\zf@check@ot@script[1]{%
429   \zf@iv@strnum{#1}%
430   \@tempcntb\XeTeXOTcountscript\zf@basefont
431   \c@zf@index\z@ \@tempswafalse
432   \loop\ifnum\c@zf@index<\@tempcntb
433     \ifnum\XeTeXOTscripttag\zf@basefont\c@zf@index=\@tempcnta
434       \@tempswatrue
435       \c@zf@index\@tempcntb
436     \else
437       \advance\c@zf@index\@ne
438     \fi
439   \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'.

```

440 \newcommand\zf@check@ot@lang[1]{%
441   \zf@iv@strnum{#1}%
442   \@tempcntb\XeTeXOTcountlanguages\zf@basefont\c@zf@script
443   \c@zf@index\z@ \@tempswafalse
444   \loop\ifnum\c@zf@index<\@tempcntb
445     \ifnum\XeTeXOTlanguage\zf@basefont\c@zf@script\c@zf@index=\@tempcnta
446       \@tempswatrue
447       \c@zf@index\@tempcntb
448     \else
449       \advance\c@zf@index\@ne
450     \fi
451   \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'.

```

452 \newcommand*\zf@check@ot@feat[1]{%
453   \@tempcntb\XeTeXOTcountfeatures\zf@basefont\c@zf@script\c@zf@language
454   \zf@v@strnum{#1}%
455   \c@zf@index\z@ \@tempswafalse
456   \loop\ifnum\c@zf@index<\@tempcntb
457     \ifnum\XeTeXOTfeaturetag\zf@basefont\c@zf@script\c@zf@language\c@zf@index=\@tempcnta
458       \@tempswatrue
459       \c@zf@index\@tempcntb
460     \else
461       \advance\c@zf@index\@ne
462     \fi
463   \repeat}
```

8.7 keyval definitions

This is the tedious section where we correlate all possible (eventually) font feature requests with their X_YTeX 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 T_EX tree. Otherwise, the argument given defines the file path of the font.

```

464 \define@key[zf]{preparse}{ExternalLocation}[]{}%
```

```

465 \zf@nobftrue\zf@noittrue
466 \def\zf@font@path{#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.

```

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

```

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

```

471 \define@key[zf]{preparse}{Script}{%
472 \edef\zf@suffix{\zf@suffix/ICU}%
473 \font\zf@basefont="\zf@fontwrap\zf@fontname\zf@suffix" at \f@size pt
474 \edef\zf@family@long{\zf@family@long +script:#1}%
475 {\setkeys[zf@feat]{Script}{#1}}}

```

Exactly the same:

```

476 \define@key[zf]{preparse}{Language}{%
477 \edef\zf@suffix{\zf@suffix/ICU}%
478 \font\zf@basefont="\zf@fontname\zf@suffix" at \f@size pt
479 \edef\zf@family@long{\zf@family@long +language:#1}%
480 {\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

```

481 \define@key[zf]{preparse}{BoldFont}{%
482 \edef\@tempa{#1}%
483 \ifx\@tempa\@empty
484 \zf@nobftrue
485 \edef\zf@family@long{\zf@family@long nobf}%
486 \else
487 \zf@nobffalse
488 \zf@partial@fontname#1\@nil
489 \let\zf@bf\@tempa
490 \edef\zf@family@long{\zf@family@long bf:#1}%
491 \fi}
492 \define@key[zf]{preparse}{ItalicFont}{%
493 \edef\@tempa{#1}%
494 \ifx\@tempa\@empty

```

```

495 \zf@noittrue
496 \edef\zf@family@long{\zf@family@long noit}%
497 \else
498 \zf@noitfalse
499 \zf@partial@fontname#1\@nil
500 \let\zf@it\@tempa
501 \edef\zf@family@long{\zf@family@long it:#1}%
502 \fi}
503 \define@key[zf]{preparse}{BoldItalicFont}{%
504 \zf@partial@fontname#1\@nil
505 \let\zf@bfit\@tempa
506 \edef\zf@family@long{\zf@family@long bfit:#1}}
507 \define@key[zf]{options}{SmallCapsFont}{%
508 \edef\@tempa{#1}%
509 \ifx\@tempa\@empty
510 \zf@nosctrue
511 \edef\zf@family@long{\zf@family@long nosc}%
512 \else
513 \zf@noscfalse
514 \zf@partial@fontname#1\@nil
515 \let\zf@sc\@tempa
516 \zf@update@family{sc:\zap@space #1 \@empty}%
517 \fi}

```

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

```

518 \def\zf@partial@fontname#1#2\@nil{%
519 \if#1*\relax
520 \edef\@tempa{\zf@fontname#2}%
521 \else
522 \edef\@tempa{#1#2}%
523 \fi}

```

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

```

524 \define@key[zf]{preparse}{UprightFeatures}{%
525 \def\zf@up@feat{,#1}%
526 \edef\zf@family@long{\zf@family@long rmfeat:#1}}
527 \define@key[zf]{preparse}{BoldFeatures}{%
528 \def\zf@bf@feat{,#1}%
529 \edef\zf@family@long{\zf@family@long bffeat:#1}}
530 \define@key[zf]{preparse}{ItalicFeatures}{%
531 \def\zf@it@feat{,#1}%
532 \edef\zf@family@long{\zf@family@long itfeat:#1}}
533 \define@key[zf]{preparse}{BoldItalicFeatures}{%
534 \def\zf@bfit@feat{,#1}%
535 \edef\zf@family@long{\zf@family@long bfitfeat:#1}}
536 \define@key[zf]{options}{SmallCapsFeatures}{%
537 \unless\ifzf@firsttime\def\zf@sc@feat{,#1}\fi
538 \zf@update@family{scfeat:\zap@space #1 \@empty}}

```

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.

```

539 \define@key[zf]{options}{Scale}{%
540   \edef\@tempa{#1}%
541   \edef\@tempb{MatchLowercase}%
542   \ifx\@tempa\@tempb
543     \zf@calc@scale{5}%
544   \else
545     \edef\@tempb{MatchUppercase}%
546     \ifx\@tempa\@tempb
547       \zf@calc@scale{8}%
548     \else
549       \edef\zf@scale{#1}%
550     \fi
551   \fi
552   \zf@update@family{+scale:\zf@scale}%
553   \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 $X_{\text{fT}}X$).

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.

```

554 \newcommand\zf@calc@scale[1]{%
555   \begingroup
556     \rmfamily
557     \setlength\@tempdima{\fontdimen#1\font}%
558     \setlength\@tempdimb{\fontdimen#1\zf@basefont}%
559     \setlength\@tempdimc{1pt*\ratio{\@tempdima}{\@tempdimb}}%
560     \xdef\zf@scale{\strip@pt\@tempdimc}%
561     \zf@PackageInfo{\zf@fontname\space scale = \zf@scale}%
562   \endgroup}

```

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

```

563 \define@key[zf]{options}{WordSpace}{%
564   \zf@update@family{+wordspace:#1}%
565   \unless\ifzf@firsttime
566     \zf@wordspace@parse#1,\zf@@ii,\zf@@iii,\zf@@
567   \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}`.

```

568 \def\zf@wordspace@parse#1,#2,#3,#4\zf@{%
569   \def\@tempa{#4}%
570   \ifx\@tempa\@empty
571     \setlength\@tempdima{#1\fontdimen2\zf@basefont}%
572     \@tempdimb\@tempdima
573     \@tempdimc\@tempdima
574   \else
575     \setlength\@tempdima{#1\fontdimen2\zf@basefont}%
576     \setlength\@tempdimb{#2\fontdimen3\zf@basefont}%
577     \setlength\@tempdimc{#3\fontdimen4\zf@basefont}%
578   \fi
579   \edef\zf@adjust{\zf@adjust
580     \fontdimen2\font\the\@tempdima
581     \fontdimen3\font\the\@tempdimb
582     \fontdimen4\font\the\@tempdimc}}

```

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

```

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

```

Letterspacing

```

587 \define@key[zf]{options}{LetterSpace}{%
588   \zf@update@family{+tracking:#1}%
589   \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.

```

590 \define@key[zf]{options}{HyphenChar}{%
591   \zf@update@family{+hyphenchar:#1}%
592   \edef\@tempa{#1}%
593   \edef\@tempb{None}%
594   \ifx\@tempa\@tempb
595     \g@addto@macro\zf@adjust{\hyphenchar\font-1\relax}%
596   \else
597     \zf@check@one@char#1\zf@@
598     \ifx\@tempb\@empty
599       {\zf@basefont\expandafter\ifnum\expandafter\XeTeXcharglyph\expandafter`#1 > 0
600         \g@addto@macro\zf@adjust{%
601           {\expandafter\hyphenchar\expandafter
602             \font\expandafter`#1}}}%
603     \else
604       \zf@PackageError
605       {\fontname\zf@basefont\space doesn't appear to have the glyph cor-
606         responding to #1.}
606       {You can't hyphenate with a character that's not available!}
607     \fi}

```

```

608 \else
609     {\zf@basefont\ifnum\XeTeXcharglyph#1 > 0
610      \g@addto@macro\zf@adjust{\hyphenchar\font#1\relax}%
611      \else
612       \zf@PackageError
613        {\fontname\zf@basefont\space doesn't appear to have the glyph cor-
        responding to #1.}
614        {You can't hyphenate with a character that's not available!}
615      \fi}
616 \fi
617 \fi}
618 \def\zf@check@one@char#1#2\zf@@{\def\@tempb{#2}}

```

Colour

```

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

```

Mapping

```

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

```

8.7.4 Continuous font axes

```

626 \define@key[zf]{options}{Weight}{%
627   \zf@update@family{+weight:#1}%
628   \zf@update@ff{weight=#1}}
629 \define@key[zf]{options}{Width}{%
630   \zf@update@family{+width:#1}%
631   \zf@update@ff{width=#1}}
632 \define@key[zf]{options}{OpticalSize}{%
633   \ifzf@icu
634     \edef\zf@suffix{\zf@suffix/S=#1}%
635     \zf@update@family{+size:#1}%
636   \fi
637   \ifzf@mm
638     \zf@update@family{+size:#1}%
639     \zf@update@ff{optical size=#1}%
640   \fi
641   \ifzf@icu\else
642     \ifzf@mm\else
643       \ifzf@firsttime
644         \zf@PackageWarning
645         {\fontname\zf@basefont\space doesn't appear to have an Opti-
        cal Size axis}%
646       \fi
647     \fi
648   \fi}

```


8.7.5 Ligatures

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

```
649 \zf@define@font@feature{Ligatures}
650 \zf@define@feature@option{Ligatures}{Required}      {1}{0}{+rlig}
651 \zf@define@feature@option{Ligatures}{NoRequired}    {1}{1}{-rlig}
652 \zf@define@feature@option{Ligatures}{Common}       {1}{2}{+liga}
653 \zf@define@feature@option{Ligatures}{NoCommon}     {1}{3}{-liga}
654 \zf@define@feature@option{Ligatures}{Rare}         {1}{4}{+dlig}
655 \zf@define@feature@option{Ligatures}{NoRare}       {1}{5}{-dlig}
656 \zf@define@feature@option{Ligatures}{Discretionary} {1}{4}{+dlig}
657 \zf@define@feature@option{Ligatures}{NoDiscretionary} {1}{5}{-dlig}
658 \zf@define@feature@option{Ligatures}{Contextual}   {}{} {+clig}
659 \zf@define@feature@option{Ligatures}{NoContextual} {}{} {-clig}
660 \zf@define@feature@option{Ligatures}{Historical}   {}{} {+hlig}
661 \zf@define@feature@option{Ligatures}{NoHistorical} {}{} {-hlig}
662 \zf@define@feature@option{Ligatures}{Logos}        {1}{6} {}
663 \zf@define@feature@option{Ligatures}{NoLogos}      {1}{7} {}
664 \zf@define@feature@option{Ligatures}{Rebus}        {1}{8} {}
665 \zf@define@feature@option{Ligatures}{NoRebus}      {1}{9} {}
666 \zf@define@feature@option{Ligatures}{Diphthong}    {1}{10} {}
667 \zf@define@feature@option{Ligatures}{NoDiphthong}  {1}{11} {}
668 \zf@define@feature@option{Ligatures}{Squared}      {1}{12} {}
669 \zf@define@feature@option{Ligatures}{NoSquared}    {1}{13} {}
670 \zf@define@feature@option{Ligatures}{AbbrevSquared} {1}{14} {}
671 \zf@define@feature@option{Ligatures}{NoAbbrevSquared} {1}{15} {}
672 \zf@define@feature@option{Ligatures}{Icelandic}    {1}{32} {}
673 \zf@define@feature@option{Ligatures}{NoIcelandic}  {1}{33} {}
```

8.7.6 Letters

```
674 \zf@define@font@feature{Letters}
675 \zf@define@feature@option{Letters}{Normal}{3}{0}{}
676 \zf@define@feature@option{Letters}{Uppercase}{3}{1}{+case}
677 \zf@define@feature@option{Letters}{Lowercase}{3}{2}{}
678 \zf@define@feature@option{Letters}{SmallCaps}{3}{3}{+smcp}
679 \zf@define@feature@option{Letters}{PetiteCaps}{}{} {+pcap}
680 \zf@define@feature@option{Letters}{UppercaseSmallCaps}{}{} {+c2sc}
681 \zf@define@feature@option{Letters}{UppercasePetiteCaps}{}{} {+c2pc}
682 \zf@define@feature@option{Letters}{InitialCaps}{3}{4}{}
683 \zf@define@feature@option{Letters}{Unicase}{}{} {+unic}
```

8.7.7 Numbers

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

Both naming conventions are offered to select the number case.

```
684 \zf@define@font@feature{Numbers}
685 \zf@define@feature@option{Numbers}{Monospaced}{6}{0}{+tnum}
```

```

686 \zf@define@feature@option{Numbers}{Proportional}{6}{1}{+pnum}
687 \zf@define@feature@option{Numbers}{Lowercase}{21}{0}{+onum}
688 \zf@define@feature@option{Numbers}{OldStyle}{21}{0}{+onum}
689 \zf@define@feature@option{Numbers}{Uppercase}{21}{1}{+lnum}
690 \zf@define@feature@option{Numbers}{Lining}{21}{1}{+lnum}
691 \zf@define@feature@option{Numbers}{SlashedZero}{14}{5}{+zero}
692 \zf@define@feature@option{Numbers}{NoSlashedZero}{14}{4}{-zero}

```

8.7.8 Contextuals

```

693 \zf@define@font@feature {Contextuals}
694 \zf@define@feature@option{Contextuals}{Swash}{}{}{+csw}
695 \zf@define@feature@option{Contextuals}{NoSwash}{}{}{-csw}
696 \zf@define@feature@option{Contextuals}{WordInitial}{8}{0}{+init}
697 \zf@define@feature@option{Contextuals}{NoWordInitial}{8}{1}{-init}
698 \zf@define@feature@option{Contextuals}{WordFinal}{8}{2}{+fina}
699 \zf@define@feature@option{Contextuals}{NoWordFinal}{8}{3}{-fina}
700 \zf@define@feature@option{Contextuals}{LineInitial}{8}{4}{}
701 \zf@define@feature@option{Contextuals}{NoLineInitial}{8}{5}{}
702 \zf@define@feature@option{Contextuals}{LineFinal}{8}{6}{+falt}
703 \zf@define@feature@option{Contextuals}{NoLineFinal}{8}{7}{-falt}
704 \zf@define@feature@option{Contextuals}{Inner}{8}{8}{+medi}
705 \zf@define@feature@option{Contextuals}{NoInner}{8}{9}{-medi}

```

8.7.9 Diacritics

```

706 \zf@define@font@feature{Diacritics}
707 \zf@define@feature@option{Diacritics}{Show}{9}{0}{}
708 \zf@define@feature@option{Diacritics}{Hide}{9}{1}{}
709 \zf@define@feature@option{Diacritics}{Decompose}{9}{2}{}

```

8.7.10 Kerning

```

710 \zf@define@font@feature{Kerning}
711 \zf@define@feature@option{Kerning}{Uppercase}{}{}{+csp}
712 \zf@define@feature@option{Kerning}{On}{}{}{+kern}
713 \zf@define@feature@option{Kerning}{Off}{}{}{-kern}
714 %\zf@define@feature@option{Kerning}{Vertical}{}{}{+vkern}
715 %\zf@define@feature@option{Kerning}{VerticalAlternateProportional}{}{}{+vpal}
716 %\zf@define@feature@option{Kerning}{VerticalAlternateHalfWidth}{}{}{+vhal}

```

8.7.11 Vertical position

```

717 \zf@define@font@feature{VerticalPosition}
718 \zf@define@feature@option{VerticalPosition}{Normal}{10}{0}{}
719 \zf@define@feature@option{VerticalPosition}{Superior}{10}{1}{+sup}
720 \zf@define@feature@option{VerticalPosition}{Inferior}{10}{2}{+sub}
721 \zf@define@feature@option{VerticalPosition}{ScientificInferior}{}{}{+sinf}
722 \zf@define@feature@option{VerticalPosition}{Ordinal}{10}{3}{+ordn}
723 \zf@define@feature@option{VerticalPosition}{Numerator}{}{}{+numr}
724 \zf@define@feature@option{VerticalPosition}{Denominator}{}{}{+dnom}

```

8.7.12 Fractions

```

725 \zf@define@font@feature{Fractions}

```

```

726 \zf@define@feature@option{Fractions}{0n}{11}{1}{+frac}
727 \zf@define@feature@option{Fractions}{Off}{11}{0}{-frac}
728 \zf@define@feature@option{Fractions}{Diagonal}{11}{2}{}
729 \zf@define@feature@option{Fractions}{Alternate}{}{}{+afrc}

```

8.7.13 Alternates and variants

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

```

730 \define@key[zf]{options}{Alternate}{%
731   \setkeys*[zf@feat]{Alternate}{#1}%
732   \unless\ifx\XKV@rm\@empty
733     \zf@make@feature{17}{#1}{}%
734   \fi}

735 \define@key[zf]{options}{Variant}{%
736   \setkeys*[zf@feat]{Variant}{#1}%
737   \unless\ifx\XKV@rm\@empty
738     \edef\@tempa{\noexpand\zf@make@feature{18}{#1}{+ss\two@digits{#1}}}\@tempa
739   \fi}

```

8.7.14 Style

```

740 \zf@define@font@feature{Style}
741 \zf@define@feature@option{Style}{Alternate}{}{}{+salt}
742 \zf@define@feature@option{Style}{Italic}{32}{2}{+ital}
743 \zf@define@feature@option{Style}{Ruby}{28}{2}{+ruby}
744 \zf@define@feature@option{Style}{Swash}{}{}{+swsh}
745 \zf@define@feature@option{Style}{Historic}{}{}{+hist}
746 \zf@define@feature@option{Style}{Display}{19}{1}{}
747 \zf@define@feature@option{Style}{Engraved}{19}{2}{}
748 \zf@define@feature@option{Style}{TitlingCaps}{19}{4}{+titl}
749 \zf@define@feature@option{Style}{TallCaps}{19}{5}{}
750 \zf@define@feature@option{Style}{HorizontalKana}{}{}{+hkna}
751 \zf@define@feature@option{Style}{VerticalKana}{}{}{+vkna}

```

8.7.15 CJK shape

```

752 \zf@define@font@feature{CJKShape}
753 \zf@define@feature@option{CJKShape}{Traditional}{20}{0}{+trad}
754 \zf@define@feature@option{CJKShape}{Simplified}{20}{1}{+smpl}
755 \zf@define@feature@option{CJKShape}{JIS1978}{20}{2}{+jp78}
756 \zf@define@feature@option{CJKShape}{JIS1983}{20}{3}{+jp83}
757 \zf@define@feature@option{CJKShape}{JIS1990}{20}{4}{+jp90}
758 \zf@define@feature@option{CJKShape}{Expert}{20}{10}{+expt}
759 \zf@define@feature@option{CJKShape}{NLC}{20}{13}{+nlck}

```

8.7.16 Character width

```

760 \zf@define@font@feature{CharacterWidth}
761 \zf@define@feature@option{CharacterWidth}{Proportional}{22}{0}{+pwid}
762 \zf@define@feature@option{CharacterWidth}{Full}{22}{1}{+fwid}
763 \zf@define@feature@option{CharacterWidth}{Half}{22}{2}{+hwid}
764 \zf@define@feature@option{CharacterWidth}{Third}{22}{3}{+twid}

```

```

765 \zf@define@feature@option{CharacterWidth}{Quarter}{22}{4}{+qwid}
766 \zf@define@feature@option{CharacterWidth}{AlternateProportional}{22}{5}{+palt}
767 \zf@define@feature@option{CharacterWidth}{AlternateHalf}{22}{6}{+halt}
768 \zf@define@feature@option{CharacterWidth}{Default}{22}{7}{+}

```

8.7.17 Annotation

```

769 \zf@define@font@feature{Annotation}
770 \zf@define@feature@option{Annotation}{Off}{24}{0}{-nalt}
771 \zf@define@feature@option{Annotation}{On}{24}{1}{+nalt}
772 \zf@define@feature@option{Annotation}{Box}{24}{1}{+}
773 \zf@define@feature@option{Annotation}{RoundedBox}{24}{2}{+}
774 \zf@define@feature@option{Annotation}{Circle}{24}{3}{+}
775 \zf@define@feature@option{Annotation}{BlackCircle}{24}{4}{+}
776 \zf@define@feature@option{Annotation}{Parenthesis}{24}{5}{+}
777 \zf@define@feature@option{Annotation}{Period}{24}{6}{+}
778 \zf@define@feature@option{Annotation}{RomanNumerals}{24}{7}{+}
779 \zf@define@feature@option{Annotation}{Diamond}{24}{8}{+}
780 \zf@define@feature@option{Annotation}{BlackSquare}{24}{9}{+}
781 \zf@define@feature@option{Annotation}{BlackRoundSquare}{24}{10}{+}
782 \zf@define@feature@option{Annotation}{DoubleCircle}{24}{11}{+}

```

8.7.18 Vertical

```

783 \zf@define@font@feature{Vertical}
784 \define@key{Zf@feat}{Vertical}{RotatedGlyphs}{\}%
785 \ifzf@icu
786 \zf@make@feature{}{}{+vrt2}%
787 \else
788 \zf@update@family{+vert}%
789 \zf@update@ff{vertical}%
790 \fi

```

8.7.19 Script

791 \newfontscript{Arabic}{arab}	\newfontscript{Armenian}{armn}
792 \newfontscript{Balinese}{bali}	\newfontscript{Bengali}{beng}
793 \newfontscript{Bopomofo}{bopo}	\newfontscript{Braille}{brai}
794 \newfontscript{Buginese}{bugi}	\newfontscript{Buhid}{buhd}
795 \newfontscript{Byzantine Music}{byzm}	\newfontscript{Canadian Syllab-
ics}{cans}	
796 \newfontscript{Cherokee}{cher}	
797 \newfontscript{CJK Ideographic}{hani}	\newfontscript{Coptic}{copt}
798 \newfontscript{Cypriot Syllabary}{cpri}	\newfontscript{Cyrillic}{cyr}
799 \newfontscript{Default}{DFLT}	\newfontscript{Deseret}{dsrt}
800 \newfontscript{Devanagari}{deva}	\newfontscript{Ethiopic}{ethi}
801 \newfontscript{Georgian}{geor}	\newfontscript{Glagolitic}{glag}
802 \newfontscript{Gothic}{goth}	\newfontscript{Greek}{grek}
803 \newfontscript{Gujarati}{gujr}	\newfontscript{Gurmukhi}{guru}
804 \newfontscript{Hangul Jamo}{jamo}	\newfontscript{Hangul}{hang}
805 \newfontscript{Hanunoo}{hano}	\newfontscript{Hebrew}{hebr}
806 \newfontscript{Hiragana and Katakana}{kana}	
807 \newfontscript{Javanese}{java}	\newfontscript{Kannada}{knda}
808 \newfontscript{Kharosthi}{khar}	\newfontscript{Khmer}{khmr}

809 \newfontscript{Lao}{lao }	\newfontscript{Latin}{latn}
810 \newfontscript{Limbu}{limb}	\newfontscript{Linear B}{linb}
811 \newfontscript{Malayalam}{mlym}	\newfontscript{Math}{math}
812 \newfontscript{Mongolian}{mong}	
813 \newfontscript{Musical Symbols}{musc}	\newfontscript{Myanmar}{mymr}
814 \newfontscript{N'ko}{nko }	\newfontscript{Ogham}{ogam}
815 \newfontscript{Old Italic}{ital}	\newfontscript{Old Persian Cuneiform}{xpeo}
816 \newfontscript{Oriya}{orya}	\newfontscript{Osmanya}{osma}
817 \newfontscript{Phags-pa}{phag}	\newfontscript{Phoenician}{phnx}
818 \newfontscript{Runic}{runr}	\newfontscript{Shavian}{shaw}
819 \newfontscript{Sinhala}{sinh}	\newfontscript{Sumero-Akkadian Cuneiform}{xsux}
820 \newfontscript{Syloti Nagri}{sylo}	\newfontscript{Syriac}{syr}
821 \newfontscript{Tagalog}{tglg}	\newfontscript{Tagbanwa}{tagb}
822 \newfontscript{Tai Le}{tale}	\newfontscript{Tai Lu}{tal}
823 \newfontscript{Tamil}{taml}	\newfontscript{Telugu}{telu}
824 \newfontscript{Thaana}{thaa}	\newfontscript{Thai}{thai}
825 \newfontscript{Tibetan}{tib}	\newfontscript{Tifinagh}{tfng}
826 \newfontscript{Ugaritic Cuneiform}{ugar}	\newfontscript{Yi}{yi }

8.7.20 Language

827 \newfontlanguage{Abaza}{ABA}	\newfontlanguage{Abkhazian}{ABK}	\newfontlanguage{Adyghe}{ADY}
828 \newfontlanguage{Afrikaans}{AFK}	\newfontlanguage{Afar}{AFR}	\newfontlanguage{Agaw}{AGW}
829 \newfontlanguage{Altai}{ALT}	\newfontlanguage{Amharic}{AMH}	\newfontlanguage{Arabic}{ARA}
830 \newfontlanguage{Aari}{ARI}	\newfontlanguage{Arakanese}{ARK}	\newfontlanguage{Assamese}{ASM}
831 \newfontlanguage{Athapaskan}{ATH}	\newfontlanguage{Avar}{AVR}	\newfontlanguage{Awadhi}{AWA}
832 \newfontlanguage{Aymara}{AYM}	\newfontlanguage{Azeri}{AZE}	\newfontlanguage{Badaga}{BAD}
833 \newfontlanguage{Baghelkhandi}{BAG}	\newfontlanguage{Balkar}{BAL}	\newfontlanguage{Baule}{BAU}
834 \newfontlanguage{Berber}{BBR}	\newfontlanguage{Bench}{BCH}	\newfontlanguage{Bible Cree}{BCR}
835 \newfontlanguage{Belarussian}{BEL}	\newfontlanguage{Bemba}{BEM}	\newfontlanguage{Bengali}{BEN}
836 \newfontlanguage{Bulgarian}{BGR}	\newfontlanguage{Bhili}{BHI}	\newfontlanguage{Bhojpuri}{BHO}
837 \newfontlanguage{Bikol}{BIK}	\newfontlanguage{Bilen}{BIL}	\newfontlanguage{Blackfoot}{BKF}
838 \newfontlanguage{Balochi}{BLI}	\newfontlanguage{Balante}{BLN}	\newfontlanguage{Balti}{BLT}
839 \newfontlanguage{Bambara}{BMB}	\newfontlanguage{Bamileke}{BML}	\newfontlanguage{Breton}{BRE}
840 \newfontlanguage{Brahui}{BRH}	\newfontlanguage{Braj Bhasha}{BRI}	\newfontlanguage{Burmese}{BRM}
841 \newfontlanguage{Bashkir}{BSH}	\newfontlanguage{Beti}{BTI}	\newfontlanguage{Catalan}{CAT}
842 \newfontlanguage{Cebuano}{CEB}	\newfontlanguage{Chechen}{CHE}	\newfontlanguage{Chaha Gurage}{CHG}
843 \newfontlanguage{Chattisgarhi}{CHH}	\newfontlanguage{Chichewa}{CHI}	\newfontlanguage{Chukchi}{CHK}
844 \newfontlanguage{Chipewyan}{CHP}	\newfontlanguage{Cherokee}{CHR}	\newfontlanguage{Chuvash}{CHU}
845 \newfontlanguage{Comorian}{CMR}	\newfontlanguage{Coptic}{COP}	\newfontlanguage{Cree}{CRE}
846 \newfontlanguage{Carrier}{CRR}	\newfontlanguage{Crimean Tatar}{CRT}	\newfontlanguage{Church Slavonic}{CSL}
847 \newfontlanguage{Czech}{CSY}	\newfontlanguage{Danish}{DAN}	\newfontlanguage{Dargwa}{DAR}
848 \newfontlanguage{Woods Cree}{DCR}	\newfontlanguage{German}{DEU}	\newfontlanguage{Default}{DFLT}
849 \newfontlanguage{Dogri}{DGR}	\newfontlanguage{Divehi}{DIV}	\newfontlanguage{Djerma}{DJR}
850 \newfontlanguage{Dangme}{DNG}	\newfontlanguage{Dinka}{DNK}	\newfontlanguage{Dungan}{DUN}
851 \newfontlanguage{Dzongkha}{DZN}	\newfontlanguage{Ebira}{EBI}	\newfontlanguage{Eastern Cree}{ECR}
852 \newfontlanguage{Edo}{EDO}	\newfontlanguage{Efik}{EFI}	\newfontlanguage{Greek}{ELL}
853 \newfontlanguage{English}{ENG}	\newfontlanguage{Erzya}{ERZ}	\newfontlanguage{Spanish}{ESP}
854 \newfontlanguage{Estonian}{ETI}	\newfontlanguage{Basque}{EUQ}	\newfontlanguage{Evenki}{EVK}
855 \newfontlanguage{Even}{EVN}	\newfontlanguage{Ewe}{EWE}	\newfontlanguage{French Antillean}{FAN}
856 \newfontlanguage{Farsi}{FAR}	\newfontlanguage{Finnish}{FIN}	\newfontlanguage{Fijian}{FJI}

857 \newfontlanguage{Flemish}{FLE}\newfontlanguage{Forest Nenets}{FNE}\newfontlanguage{Fon}{FON}
858 \newfontlanguage{Faroese}{FOS}\newfontlanguage{French}{FRA}\newfontlanguage{Frisian}{FRI}
859 \newfontlanguage{Friulian}{FRL}\newfontlanguage{Futa}{FTA}\newfontlanguage{Fulani}{FUL}
860 \newfontlanguage{Ga}{GAD}\newfontlanguage{Gaelic}{GAE}\newfontlanguage{Gagauz}{GAG}
861 \newfontlanguage{Galician}{GAL}\newfontlanguage{Garshuni}{GAR}\newfontlanguage{Garhwali}{GAW}
862 \newfontlanguage{Ge'ez}{GEZ}\newfontlanguage{Gilyak}{GIL}\newfontlanguage{Gumuz}{GMZ}
863 \newfontlanguage{Gondi}{GON}\newfontlanguage{Greenlandic}{GRN}\newfontlanguage{Garó}{GRO}
864 \newfontlanguage{Guarani}{GUA}\newfontlanguage{Gujarati}{GUJ}\newfontlanguage{Haitian}{HAI}
865 \newfontlanguage{Halam}{HAL}\newfontlanguage{Harauti}{HAR}\newfontlanguage{Hausa}{HAU}
866 \newfontlanguage{Hawain}{HAW}\newfontlanguage{Hammer-Banna}{HBN}\newfontlanguage{Hiligaynon}{HIL}
867 \newfontlanguage{Hindi}{HIN}\newfontlanguage{High Mari}{HMA}\newfontlanguage{Hindko}{HND}
868 \newfontlanguage{Ho}{HO}\newfontlanguage{Harari}{HRI}\newfontlanguage{Croatian}{HRV}
869 \newfontlanguage{Hungarian}{HUN}\newfontlanguage{Armenian}{HYE}\newfontlanguage{Igbo}{IBO}
870 \newfontlanguage{Ijo}{IJO}\newfontlanguage{Ilokano}{ILO}\newfontlanguage{Indonesian}{IND}
871 \newfontlanguage{Ingush}{ING}\newfontlanguage{Inuktitut}{INU}\newfontlanguage{Irish}{IRI}
872 \newfontlanguage{Irish Traditional}{IRT}\newfontlanguage{Icelandic}{ISL}\newfontlanguage{Inari Sami}{INR}
873 \newfontlanguage{Italian}{ITA}\newfontlanguage{Hebrew}{IWR}\newfontlanguage{Javanese}{JAV}
874 \newfontlanguage{Yiddish}{JII}\newfontlanguage{Japanese}{JAN}\newfontlanguage{Judezmo}{JUD}
875 \newfontlanguage{Jula}{JUL}\newfontlanguage{Kabardian}{KAB}\newfontlanguage{Kachchi}{KAC}
876 \newfontlanguage{Kalenjin}{KAL}\newfontlanguage{Kannada}{KAN}\newfontlanguage{Karachay}{KAR}
877 \newfontlanguage{Georgian}{KAT}\newfontlanguage{Kazakh}{KAZ}\newfontlanguage{Kebena}{KEB}
878 \newfontlanguage{Khutsuri Georgian}{KGE}\newfontlanguage{Khakass}{KHA}\newfontlanguage{Khanty-Kazim}{KHK}
879 \newfontlanguage{Khmer}{KHM}\newfontlanguage{Khanty-Shurishkar}{KHS}\newfontlanguage{Khanty-Vakhi}{KHV}
880 \newfontlanguage{Khowar}{KHW}\newfontlanguage{Kikuyu}{KIK}\newfontlanguage{Kirghiz}{KIR}
881 \newfontlanguage{Kisii}{KIS}\newfontlanguage{Kokni}{KKN}\newfontlanguage{Kalmyk}{KLM}
882 \newfontlanguage{Kamba}{KMB}\newfontlanguage{Kumaoni}{KMN}\newfontlanguage{Komo}{KMO}
883 \newfontlanguage{Komso}{KMS}\newfontlanguage{Kanuri}{KNR}\newfontlanguage{Kodagu}{KOD}
884 \newfontlanguage{Korean Old Hangul}{KOH}\newfontlanguage{Konkani}{KOK}\newfontlanguage{Kikongo}{KIKO}
885 \newfontlanguage{Komi-Permyak}{KOP}\newfontlanguage{Korean}{KOR}\newfontlanguage{Komi-Zyrian}{KOZ}
886 \newfontlanguage{Kpelle}{KPL}\newfontlanguage{Krio}{KRI}\newfontlanguage{Karakalpak}{KPK}
887 \newfontlanguage{Karelian}{KRL}\newfontlanguage{Karaim}{KRM}\newfontlanguage{Karen}{KRN}
888 \newfontlanguage{Koorote}{KRT}\newfontlanguage{Kashmiri}{KSH}\newfontlanguage{Khasi}{KSI}
889 \newfontlanguage{Kildin Sami}{KSM}\newfontlanguage{Kui}{KUI}\newfontlanguage{Kulvi}{KUL}
890 \newfontlanguage{Kumyk}{KUM}\newfontlanguage{Kurdish}{KUR}\newfontlanguage{Kurukh}{KUU}
891 \newfontlanguage{Kuy}{KUY}\newfontlanguage{Koryak}{KYK}\newfontlanguage{Ladin}{LAD}
892 \newfontlanguage{Lahuli}{LAH}\newfontlanguage{Lak}{LAK}\newfontlanguage{Lambani}{LAM}
893 \newfontlanguage{Lao}{LAO}\newfontlanguage{Latin}{LAT}\newfontlanguage{Laz}{LAZ}
894 \newfontlanguage{L-Cree}{LCR}\newfontlanguage{Ladakhi}{LDK}\newfontlanguage{Lezgi}{LEZ}
895 \newfontlanguage{Lingala}{LIN}\newfontlanguage{Low Mari}{LMA}\newfontlanguage{Limbu}{LMB}
896 \newfontlanguage{Lomwe}{LMW}\newfontlanguage{Lower Sorbian}{LSB}\newfontlanguage{Lule Sami}{LSM}
897 \newfontlanguage{Lithuanian}{LTH}\newfontlanguage{Luba}{LUB}\newfontlanguage{Luganda}{LUG}
898 \newfontlanguage{Luhya}{LUH}\newfontlanguage{Luo}{LUO}\newfontlanguage{Latvian}{LVI}
899 \newfontlanguage{Majang}{MAJ}\newfontlanguage{Makua}{MAK}\newfontlanguage{Malayalam Traditional}{MAL}
900 \newfontlanguage{Mansi}{MAN}\newfontlanguage{Marathi}{MAR}\newfontlanguage{Marwari}{MAW}
901 \newfontlanguage{Mbundu}{MBN}\newfontlanguage{Manchu}{MCH}\newfontlanguage{Moose Cree}{MCR}
902 \newfontlanguage{Mende}{MDE}\newfontlanguage{Me'en}{MEN}\newfontlanguage{Mizo}{MIZ}
903 \newfontlanguage{Macedonian}{MKD}\newfontlanguage{Male}{MLE}\newfontlanguage{Malagasy}{MLG}

904 \newfontlanguage{Malinke}{MLN} \newfontlanguage{Malayalam Reformed}{MLR} \newfontlanguage{Malay}{MAY}
 905 \newfontlanguage{Mandinka}{MND} \newfontlanguage{Mongolian}{MNG} \newfontlanguage{Manipuri}{MNI}
 906 \newfontlanguage{Maninka}{MNK} \newfontlanguage{Manx Gaelic}{MNX} \newfontlanguage{Moksha}{MOK}
 907 \newfontlanguage{Moldavian}{MOL} \newfontlanguage{Mon}{MON} \newfontlanguage{Moroccan}{MOR}
 908 \newfontlanguage{Maori}{MRI} \newfontlanguage{Maithili}{MTH} \newfontlanguage{Maltese}{MTS}
 909 \newfontlanguage{Mundari}{MUN} \newfontlanguage{Naga-Assamese}{NAG} \newfontlanguage{Nanai}{NAN}
 910 \newfontlanguage{Naskapi}{NAS} \newfontlanguage{N-Cree}{NCR} \newfontlanguage{Ndebele}{NDB}
 911 \newfontlanguage{Ndonga}{NDG} \newfontlanguage{Nepali}{NEP} \newfontlanguage{Newari}{NEW}
 912 \newfontlanguage{Nagari}{NGR} \newfontlanguage{Norway House Cree}{NHC} \newfontlanguage{Nisi}{NIS}
 913 \newfontlanguage{Niuean}{NIU} \newfontlanguage{Nkole}{NKL} \newfontlanguage{N'ko}{NKO}
 914 \newfontlanguage{Dutch}{NLD} \newfontlanguage{Nogai}{NOG} \newfontlanguage{Norwegian}{NOR}
 915 \newfontlanguage{Northern Sami}{NSM} \newfontlanguage{Northern Tai}{NTA} \newfontlanguage{Esperanto}{NTO}
 916 \newfontlanguage{Nynorsk}{NYN} \newfontlanguage{Oji-Cree}{OCR} \newfontlanguage{Ojibway}{OJB}
 917 \newfontlanguage{Oriya}{ORI} \newfontlanguage{Oromo}{ORO} \newfontlanguage{Ossetian}{OSS}
 918 \newfontlanguage{Palestinian Aramaic}{PAA} \newfontlanguage{Pali}{PAL} \newfontlanguage{Punjabi}{PAN}
 919 \newfontlanguage{Palpa}{PAP} \newfontlanguage{Pashto}{PAS} \newfontlanguage{Polytonic Greek}{PGR}
 920 \newfontlanguage{Pilipino}{PIL} \newfontlanguage{Palaung}{PLG} \newfontlanguage{Polish}{PLK}
 921 \newfontlanguage{Provençal}{PRO} \newfontlanguage{Portuguese}{PTG} \newfontlanguage{Chin}{QIN}
 922 \newfontlanguage{Rajasthani}{RAJ} \newfontlanguage{R-Cree}{RCR} \newfontlanguage{Russian Buriat}{RBU}
 923 \newfontlanguage{Riang}{RIA} \newfontlanguage{Rhaeto-Romanic}{RMS} \newfontlanguage{Romanian}{ROM}
 924 \newfontlanguage{Romany}{ROY} \newfontlanguage{Rusyn}{RSY} \newfontlanguage{Ruanda}{RUA}
 925 \newfontlanguage{Russian}{RUS} \newfontlanguage{Sadri}{SAD} \newfontlanguage{Sanskrit}{SAN}
 926 \newfontlanguage{Santali}{SAT} \newfontlanguage{Sayisi}{SAY} \newfontlanguage{Sekota}{SEK}
 927 \newfontlanguage{Selkup}{SEL} \newfontlanguage{Sango}{SGO} \newfontlanguage{Shan}{SHN}
 928 \newfontlanguage{Sibe}{SIB} \newfontlanguage{Sidamo}{SID} \newfontlanguage{Silte Gurage}{SIG}
 929 \newfontlanguage{Skolt Sami}{SKS} \newfontlanguage{Slovak}{SKY} \newfontlanguage{Slavey}{SLA}
 930 \newfontlanguage{Slovenian}{SLV} \newfontlanguage{Somali}{SML} \newfontlanguage{Samoan}{SMO}
 931 \newfontlanguage{Sena}{SNA} \newfontlanguage{Sindhi}{SND} \newfontlanguage{Sinhalese}{SNH}
 932 \newfontlanguage{Soninke}{SNK} \newfontlanguage{Sodo Gurage}{SOG} \newfontlanguage{Sotho}{SOT}
 933 \newfontlanguage{Albanian}{SQI} \newfontlanguage{Serbian}{SRB} \newfontlanguage{Saraiki}{SRK}
 934 \newfontlanguage{Serer}{SRR} \newfontlanguage{South Slavey}{SSL} \newfontlanguage{Southern Sami}{SSM}
 935 \newfontlanguage{Suri}{SUR} \newfontlanguage{Svan}{SVA} \newfontlanguage{Swedish}{SVE}
 936 \newfontlanguage{Swadaya Aramaic}{SWA} \newfontlanguage{Swahili}{SWK} \newfontlanguage{Swazi}{SWZ}
 937 \newfontlanguage{Sutu}{SXT} \newfontlanguage{Syriac}{SYR} \newfontlanguage{Tabasaran}{TAB}
 938 \newfontlanguage{Tajiki}{TAJ} \newfontlanguage{Tamil}{TAM} \newfontlanguage{Tatar}{TAT}
 939 \newfontlanguage{TH-Cree}{TCR} \newfontlanguage{Telugu}{TEL} \newfontlanguage{Tongan}{TGN}

```

940 \newfontlanguage{Tigre}{TGR} \newfontlanguage{Tigrinya}{TGY} \newfontlanguage{Thai}{THA}
941 \newfontlanguage{Tahitian}{THT} \newfontlanguage{Tibetan}{TIB} \newfont-
    language{Turkmen}{TKM}
942 \newfontlanguage{Temne}{TMN} \newfontlanguage{Tswana}{TNA} \newfontlanguage{Tundra Nenets}{TNE}
943 \newfontlanguage{Tonga}{TNG} \newfontlanguage{Todo}{TOD}
944 \newfontlanguage{Tsonga}{TSG} \newfontlanguage{Turoyo Aramaic}{TUA} \new-
    fontlanguage{Tulu}{TUL}
945 \newfontlanguage{Tuvina}{TUV} \newfontlanguage{Twana}{TWI} \newfontlanguage{Udmurt}{UDM}
946 \newfontlanguage{Ukrainian}{UKR} \newfontlanguage{Urdu}{URD} \newfontlanguage{Upper Sor-
    bian}{USB}
947 \newfontlanguage{Uyghur}{UYG} \newfontlanguage{Uzbek}{UZB} \newfontlanguage{Venda}{VEN}
948 \newfontlanguage{Vietnamese}{VIT} \newfontlanguage{Wa}{WA} \newfontlanguage{Wagdi}{WAG}
949 \newfontlanguage{West-Cree}{WCR} \newfontlanguage{Welsh}{WEL} \newfontlanguage{Wolof}{WLF}
950 \newfontlanguage{Tai Lue}{XBD} \newfontlanguage{Xhosa}{XHS} \newfontlanguage{Yakut}{YAK}
951 \newfontlanguage{Yoruba}{YBA} \newfontlanguage{Y-Cree}{YCR} \newfontlanguage{Yi Clas-
    sic}{YIC}
952 \newfontlanguage{Yi Modern}{YIM} \newfontlanguage{Chinese Hong Kong}{ZHH}
953 \newfontlanguage{Chinese Phonetic}{ZHP} \newfontlanguage{Chinese Simpli-
    fied}{ZHS}
954 \newfontlanguage{Chinese Traditional}{ZHT} \newfontlanguage{Zande}{ZND} \new-
    fontlanguage{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:

```

955 \define@key[zf@feat]{Lang}{Turkish}[]{}%
956 \zf@check@ot@lang{TRK}%
957 \if@tempswa
958 \c@zf@language\@tempcnta\relax
959 \xdef\zf@language@name{Turkish}%
960 \xdef\zf@family@long{\zf@family@long+lang=Turkish}%
961 \xdef\zf@pre@ff{\zf@pre@ff language=TRK,}%
962 \else
963 \zf@check@ot@lang{TUR}%
964 \if@tempswa
965 \c@zf@language\@tempcnta\relax
966 \xdef\zf@language@name{Turkish}%
967 \xdef\zf@family@long{\zf@family@long+lang=Turkish}%
968 \xdef\zf@pre@ff{\zf@pre@ff language=TUR,}%
969 \else
970 \zf@PackageWarning{Font \fontname\zf@basefont does not contain
971 language '#1' for script '\zf@script@name'}%
972 \fi
973 \fi}

```

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` 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.

```

974 \providecommand*\sidefault{si}
975 \DeclareRobustCommand{\sishape}{%
976   \not@math@alphabet\sishape\relax
977   \fontshape\sidefault\selectfont}
978 \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.

```

979 \newcommand*\zf@merge@shape}[3]{%
980   \edef\@tempa{#1}%
981   \edef\@tempb{#2}%
982   \ifx\@tempa\@tempb
983     \ifcsname\@tempa\encoding/\@tempa/family/\@tempa/series/#3\endcsname
984       \edef\@tempa{#3}%
985     \fi
986   \fi
987   \fontshape{\@tempa}\selectfont}

```

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

`\scshape`

`\upshape`

```

988 \DeclareRobustCommand{\itshape}{%
989   \not@math@alphabet\itshape\mathit
990   \zf@merge@shape\itdefault\scdefault\sidefault}
991 \DeclareRobustCommand{\slshape}{%
992   \not@math@alphabet\slshape\relax
993   \zf@merge@shape\sldefault\scdefault\sidefault}
994 \DeclareRobustCommand{\scshape}{%
995   \not@math@alphabet\scshape\relax
996   \zf@merge@shape\scdefault\itdefault\sidefault}
997 \DeclareRobustCommand{\upshape}{%
998   \not@math@alphabet\upshape\relax
999   \zf@merge@shape\updefault\sidefault\scdefault}

```

`\em` Redefinitions moved to the `xltextra` package.

`\emph`

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 `\setromanfont` and `\friends` was run.

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

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

```

1000 \@ifpackageloaded{euler}{\zf@package@euler@loadedtrue}
1001                          {\zf@package@euler@loadedfalse}
1002 \AtBeginDocument{%
1003   \let\zf@font@warning\@font@warning
1004   \let\@font@warning\@font@info
1005   \@ifpackageloaded{euler}{%
1006     \ifzf@package@euler@loaded
1007       \zf@math@eulertrue
1008     \else
1009       \zf@PackageError{The euler package must be loaded BEFORE fontspec}
1010       {fontspec only overwrites euler's attempt to \MessageBreak
1011         define the maths text fonts if fontspec is \MessageBreak
1012         loaded after euler. Type <return> to proceed\MessageBreak
1013         with incorrect \protect\mathit, \protect\mathbf, etc}
1014       \fi}{%
1015     \@ifpackageloaded{lucbmath}{\zf@math@lucidatrue}{%
1016       \@ifpackageloaded{lucidabr}{\zf@math@lucidatrue}{%
1017         \@ifpackageloaded{lucimatx}{\zf@math@lucidatrue}{%

```

Knuth's CM 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 L^AT_EX'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 Euler-Fraktur, but it's *ugly*. So I ignore it. Sorry if this causes inconvenience.)

```

1018 \DeclareSymbolFont{legacymaths}{OT1}{cmr}{m}{n}
1019 \SetSymbolFont{legacymaths}{bold}{OT1}{cmr}{bx}{n}
1020 \DeclareMathAccent{\acute}{\mathalpha}{legacymaths}{19}
1021 \DeclareMathAccent{\grave}{\mathalpha}{legacymaths}{18}
1022 \DeclareMathAccent{\ddot}{\mathalpha}{legacymaths}{127}
1023 \DeclareMathAccent{\tilde}{\mathalpha}{legacymaths}{126}
1024 \DeclareMathAccent{\bar}{\mathalpha}{legacymaths}{22}
1025 \DeclareMathAccent{\breve}{\mathalpha}{legacymaths}{21}
1026 \DeclareMathAccent{\check}{\mathalpha}{legacymaths}{20}
1027 \DeclareMathAccent{\hat}{\mathalpha}{legacymaths}{94} % too bad, euler
1028 \DeclareMathAccent{\dot}{\mathalpha}{legacymaths}{95}
1029 \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{:}\mskip6mu\plus1mu\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}
```

(3A₁₆ = 58₁₀) 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.

```
1030 \begingroup
1031 \mathchardef\@tempa="603A %
1032 \let\next\egroup
1033 \ifx\colon\@tempa
1034 \DeclareMathSymbol{\colon}{\mathpunct}{legacymaths}{58}
1035 \fi
1036 \endgroup
```

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

```
1037 \ifzf@math@euler\else
1038 \DeclareMathSymbol{!}{\mathclose}{legacymaths}{33}
1039 \DeclareMathSymbol{:}{\mathrel}{legacymaths}{58}
1040 \DeclareMathSymbol{;}{\mathpunct}{legacymaths}{59}
1041 \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.

```
1042 \ifzf@math@lucida\else
1043 \DeclareMathSymbol{0}{\mathalpha}{legacymaths}{`0}
1044 \DeclareMathSymbol{1}{\mathalpha}{legacymaths}{`1}
1045 \DeclareMathSymbol{2}{\mathalpha}{legacymaths}{`2}
1046 \DeclareMathSymbol{3}{\mathalpha}{legacymaths}{`3}
1047 \DeclareMathSymbol{4}{\mathalpha}{legacymaths}{`4}
1048 \DeclareMathSymbol{5}{\mathalpha}{legacymaths}{`5}
1049 \DeclareMathSymbol{6}{\mathalpha}{legacymaths}{`6}
1050 \DeclareMathSymbol{7}{\mathalpha}{legacymaths}{`7}
1051 \DeclareMathSymbol{8}{\mathalpha}{legacymaths}{`8}
1052 \DeclareMathSymbol{9}{\mathalpha}{legacymaths}{`9}
1053 \DeclareMathSymbol{\Gamma}{\mathalpha}{legacymaths}{0}
1054 \DeclareMathSymbol{\Delta}{\mathalpha}{legacymaths}{1}
1055 \DeclareMathSymbol{\Theta}{\mathalpha}{legacymaths}{2}
```

```

1056 \DeclareMathSymbol{\Lambda}{\mathalpha}{legacymaths}{3}
1057 \DeclareMathSymbol{\Xi}{\mathalpha}{legacymaths}{4}
1058 \DeclareMathSymbol{\Pi}{\mathalpha}{legacymaths}{5}
1059 \DeclareMathSymbol{\Sigma}{\mathalpha}{legacymaths}{6}
1060 \DeclareMathSymbol{\Upsilon}{\mathalpha}{legacymaths}{7}
1061 \DeclareMathSymbol{\Phi}{\mathalpha}{legacymaths}{8}
1062 \DeclareMathSymbol{\Psi}{\mathalpha}{legacymaths}{9}
1063 \DeclareMathSymbol{\Omega}{\mathalpha}{legacymaths}{10}
1064 \DeclareMathSymbol{+}{\mathbin}{legacymaths}{43}
1065 \DeclareMathSymbol{=}{\mathrel}{legacymaths}{61}
1066 \DeclareMathDelimiter{\}{\mathopen}{legacymaths}{40}{largesymbols}{0}
1067 \DeclareMathDelimiter{\)}{\mathclose}{legacymaths}{41}{largesymbols}{1}
1068 \DeclareMathDelimiter{\[}{\mathopen}{legacymaths}{91}{largesymbols}{2}
1069 \DeclareMathDelimiter{\]}{\mathclose}{legacymaths}{93}{largesymbols}{3}
1070 \DeclareMathDelimiter{/}{\mathord}{legacymaths}{47}{largesymbols}{14}
1071 \DeclareMathSymbol{\mathdollar}{\mathord}{legacymaths}{36}
1072 \fi
1073 \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 L^AT_EX 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`.

```

1074 \DeclareSymbolFont{operators}\zf@enc\zf@rmmaths\mddefault\updefault
1075 \SetSymbolFont{operators}{normal}\zf@enc\zf@rmmaths\mddefault\updefault
1076 \SetMathAlphabet\mathrm{normal}\zf@enc\zf@rmmaths\mddefault\updefault
1077 \SetMathAlphabet\mathit{normal}\zf@enc\zf@rmmaths\mddefault\itdefault
1078 \SetMathAlphabet\mathbf{normal}\zf@enc\zf@rmmaths\bfdefault\updefault
1079 \SetMathAlphabet\mathsf{normal}\zf@enc\zf@sfmaths\mddefault\updefault
1080 \SetMathAlphabet\mathtt{normal}\zf@enc\zf@ttmaths\mddefault\updefault
1081 \SetSymbolFont{operators}{bold}\zf@enc\zf@rmmaths\bfdefault\updefault
1082 \ifdefined\zf@rmboldmaths
1083 \SetMathAlphabet\mathrm{bold}\zf@enc\zf@rmboldmaths\mddefault\updefault
1084 \SetMathAlphabet\mathbf{bold}\zf@enc\zf@rmboldmaths\bfdefault\updefault
1085 \SetMathAlphabet\mathit{bold}\zf@enc\zf@rmboldmaths\mddefault\itdefault
1086 \else
1087 \SetMathAlphabet\mathrm{bold}\zf@enc\zf@rmmaths\bfdefault\updefault
1088 \SetMathAlphabet\mathit{bold}\zf@enc\zf@rmmaths\bfdefault\itdefault
1089 \fi
1090 \SetMathAlphabet\mathsf{bold}\zf@enc\zf@sfmaths\bfdefault\updefault
1091 \SetMathAlphabet\mathtt{bold}\zf@enc\zf@ttmaths\bfdefault\updefault
1092 \let\font@warning\zf@font@warning}

```

8.10 Finishing up

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

```

1093 \if@zf@configfile
1094 \InputIfFileExists{fontspec.cfg}
1095 {\typeout{fontspec.cfg loaded.}}

```

1096 {\typeout{No fontspec.cfg file found; no configuration loaded.}}

1097 \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.

```
1
2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3 %%% FOR BACKWARDS COMPATIBILITY WITH PREVIOUS VERSIONS %%%
4
5 \let\newfontinstance\newfontfamily
6
7 \newcommand\newfeaturecode[3]{%
8   \define@key{zf}{#1}[]{\zf@make@feature{#2}{#3}}}%
9
10 \aliasfontfeature{BoldFont}{Bold}
11 \aliasfontfeature{ItalicFont}{Italic}
12 \aliasfontfeature{BoldItalicFont}{BoldItalic}
13 \aliasfontfeature{SmallCapsFont}{SmallCaps}
14 \aliasfontfeature{Style}{StyleOptions}
15 \aliasfontfeature{Contextuals}{Swashes}
16 \aliasfontfeatureoption{Contextuals}{Swash}{Contextual}
17 \aliasfontfeatureoption{Letters}{UppercaseSmallCaps}{SMALLCAPS}
18 \aliasfontfeatureoption{Letters}{UppercasePetiteCaps}{PETITECAPS}
19
20 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
21 %%% FOR CONVENIENCE %%%
22
23 \newfontscript{Kana}{kana}
24 \newfontscript{Maths}{math}
25 \newfontscript{CJK}{hani}
26
```

File III

fontspec-example.ltx

```
1 \documentclass{article}
2
3 \usepackage{euler}
4 \usepackage[cm-default]{fontspec}
5 \usepackage{xltextra}
6
7 \defaultfontfeatures{Scale=MatchLowercase,Mapping=tex-text}
8 \setromanfont[Numbers=Lowercase]{FPL Neu}
9 \setsansfont{Lucida Sans}
10 \setmonofont{Lucida Sans Typewriter}
11
12 \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 for \La-
    TeX{} documents typeset with \XeTeX{}. The basic command is\
20 \indent \verb!\fontspec[font features]{font display name}!\
21 As an example:
22
23 \begin{center}
24   \Large
25   \fontspec[
26     Colour          = 0000CC,
27     Numbers          = OldStyle,
28     VerticalPosition = Ordinal,
29     Variant          = 2
30   ]{Apple Chancery}
31   My 1st example of Apple Chancery
32 \end{center}
33
34 The default roman, sans serif, and typewriter fonts may be set with the \verb!\setromanfont!, \verb!
    \setseriffont!, and \verb!\setmonospacefont! commands, respectively, as shown in the preamble. They take the same syn-
    tax as the \verb!\fontspec! package. All expected font shapes are available:
35
36 \begin{center}
37   {\itshape Italics and \scshape small caps\dots}\
38   {\sffamily\bfseries Bold sans serif and \itshape bold italic sans serif\dots}
39 \end{center}
40
41 With the roman and sans serif fonts set in the preamble, text fonts in math mode are also changed: $
    face 'Euler' has been used in this document (with the \textsf{euler} pack-
    age---or the \textsf{eulervm} package if the lxdvdimx driver is be-
    ing used), since the default Computer Modern maths font is rather light.
42 \[
43   \mathcal{F}(s) = \int_0^\infty f(t) \exp(-st) \mathrm{d}t
44 \]
45
46 You'll also notice the \verb!\defaultfontfeatures! command in the pream-
    ble. This command takes a single argument of font features that are then ap-
    plied to every subsequent instance of font selection. The first argu-
    ment in this case, \verb!\Mapping=tex-text!, enables regular \TeX{} liga-
    tures like \verb!`---'! for '---'. The second automatically scales the fonts to the same x-
    height.
47
48 Please see the documentation for font feature explanation and further pack-
    age niceties.
49
50 \end{document}

```

Change History

v1.0	
General: Initial version.	27
v1.1	
General: Name change to fontspec.	27
\setromanfont: Implemented (with friends).	29
v1.10	
General: Color brought back into the .sty	47
New feature LetterSpace.	46
Some babel encoding problems resolved.	28
\addfontfeatures: Saved family information macro changes.	31
\AtBeginDocument: Added lucimatx checking. (Not really tested, though.)	56
Fixed Lucida bug (missing \else)	56
\zf@fontspec: Saved family info split into two (now three) macros.	36
Space zapped from L ^A T _E X family name due to various problems.	36
\zf@make@feature: Removed embarrassing space after warnings.	39
v1.11	
General: HyphenChar checks its input now.	46
Added better support for Turkish language selection.	55
Ensure bold/italic fonts are loaded with the same renderer as the regular font even if unspecified.	36
OpenType Variant fixed.	50
\emph: Redefinitions moved to xltextra.	56
\newfontface: Name change from \newfontfamily.	30
\newfontlanguage: Fixed \c@zf@language setting not being global.	33
\newfontscript: Fixed \c@zf@script setting not being global.	32
\zf@wordspace@parse: Improved saving \fontdimen stuff to \zd@adjust(also see PunctuationSpace).	46
v1.12	
General: BoldFont, etc., flags \zf@nobf conditional false rather than assuming it implicitly. This allows, e.g., empty BoldFont to be overloaded.	43
Finally, use the EU1 font encoding (and the LM fonts).	28
New feature ExternalLocation for loading external fonts.	43
Package option for disabling the EU1 encoding.	28
\addfontfeatures: Now use grouping to restore \zf@default@option change.	31
\zf@make@aat@feature@string: Fixed result of \XeTeXfeaturename output change (empty string if odd non-exclusive selector).	41
Removed \@thisfontfeature macro; replaced with \@tempa.	41
v1.2	
General: Initial OpenType support.	27
Support for Scale.	45
v1.3	
General: More OpenType support.	27
Support for Mapping and Colour.	47
\defaultfontfeatures: Implemented.	30
\newAATfeature: Implemented.	32
\newfontfeature: Implemented.	31
v1.3a	
General: Bug fix for OpenType small caps.	48
v1.4	
General: Support for Weight and Width AAT features.	47

\AtBeginDocument: Selects the default \mathXX fonts.	56
\defaultfontfeatures: Name changed from \setdefaultoptions.	30
v1.5	
General: New options for arbitrary bold/italic shapes.	43
\addfontfeatures: Implemented.	31
\zf@fontspec: Added code for choosing arbitrary bold/italic fonts.	36
Checks if the font family has already been defined.	36
NFSS specifiers now take the default values.	36
\zf@make@font@shapes: Absorbed font-checking from \zf@fontspec.	37
v1.5a	
\AtBeginDocument: Added fix for Computer Modern maths.	56
v1.6	
General: Bold option aliased to BoldFont.	43
LetterCase is now Letters and options changed appropriately.	48
Scale feature now updates family name.	45
All AAT Fractions features offered.	49
New OpenType feature: Language	52
New OpenType feature: Script	51
OpenType letters features: PetiteCaps and PETITECAPS.	48
OpenType ligature features: Contextual and Historical.	48
OpenType stylistic sets supports under the Variant option.	50
\addfontfeatures: Removed \relaxing of temporary macros.	31
\AtBeginDocument: Removed mathtime support since XeTeX doesn't handle	
virtual fonts. Why did I put it in in the first place?	56
\fontspec: Removed \zf@currfont (unnecessary)	29
\newfontface: Implemented.	30
\newfontfeature: newff counter now uses LaTeX methods rather than prim-	
itive TeX. I don't know if there is any advantage to this.	31
\setromanfont: Changed \rmdefault, etc., assigning to use \let directly.	29
\zf@fontspec: Added code for choosing arbitrary bold/italic font features.	36
Writes some info to the .log file	36
\zf@get@feature@requests: Removed the space between the comma and	
\zf@options when it's concatenated with the defaults.	38
v1.7	
General: Style feature renamed from StyleOptions.	50
AAT Numbers:SlashedZero.	48
New feature: Annotation	51
New feature: CharacterShape	50
New feature: CharacterWidth	50
New feature: OpticalSize; works with both OpenType and MM fonts.	47
OpenType Alternate Fractions feature.	49
OpenType Alternate now only AAT.	50
Removed AAT check for weight/width axes (could also be Multiple Master)	47
\zf@define@feature@option: Implemented for the bulk of the feature pro-	
cessing code.	39
\zf@fontspec: Optional argument now mandatory.	36
\zf@make@aat@feature@string: Changed some \edefs to \let	41
Removed third argument; always saves the feature string in \zf@thisfontfeature	41
\zf@make@feature: Accommodation of the \zf@thisfontfeature change.	39
\zf@make@font@shapes: Changed some \edefs to \let.	37
Support for the OpticalSize feature.	37

\zf@make@smallcaps: Accommodation of the \zf@thisfontfeature change.	39
\zf@set@font@type: Added ‘MM’ font type; tests true, <i>e.g.</i> , with Skia & Minion MM. Used with the OpticalSize feature.	36
Removed exclusivity from font type (AAT, OpenType) check, since fonts can be both.	36
Removed various \count255s.	36
\zf@update@ff: Fix for featureless fonts (<i>e.g.</i> , the MS fonts) being ignored.	39
v1.8	
\AtBeginDocument: Added support for user-specified \mathrm and others.	56
Finally fixed legacy maths font issues. Also checks that euler.sty is loaded in the right order.	56
\setmathrm: Implemented (with friends).	30
v1.8a	
\AtBeginDocument: Added conditional to \colon math symbol (incompatibility with lucida and amsmath)	56
v1.9	
General: CharacterShape now CJKShape	50
SMALLCAPS option changed to UppercaseSmallCaps to facilitate option normalisation (to come). Similarly for PETITECAPS.	48
Swashes feature changed to Contextuals. Option of this feature Contextual changed to Swash, for obvious reasons.	49
TextSpacing now CharacterWidth, with associated option names’ change.	50
Alternate/Variant options can be assigned names.	50
New Scale options: MatchLowercase and MatchUppercase.	45
New feature HyphenChar.	46
New feature Kerning.	49
New feature PunctuationSpace.	46
New feature UprightFeatures.	43
New feature Vertical.	51
New feature WordSpace.	45
New features SmallCapsFont and SmallCapsFeatures.	43
Package options (no)config, quiet implemented.	60
\addfontfeatures: Added \ignorespaces to make it invisible.	31
Changed \fontspec call to \@fontspec so that \ignorespaces isn’t called unnecessarily.	31
\aliasfontfeature: Implemented.	32
\aliasfontfeatureoption: Implemented.	32
\AtBeginDocument: Maths hex numbers converted to decimal.	56
Suppresses harmless maths font encoding warnings.	59
\emph: Redefined \em in order for nested emphases to work.	56
\fontspec: Added \ignorespaces to make it invisible.	29
\keyval@alias@key: Implemented.	40
\multi@alias@key: Implemented for \aliasfontfeature.	40
\newAATfeature: Replacement for \newfeaturecode.	32
\newfontlanguage: Implemented.	33
\newfontscript: Implemented.	32
\newICUfeature: Implemented.	32
\zf@calc@scale: Implemented for auto-scaling options.	45
\zf@check@ot@feat: Implemented.	42
\zf@check@ot@lang: Implemented.	42
\zf@check@ot@script: Implemented.	41

\zf@DeclareFontShape: Implemented as wrapper for \DeclareFontShape.	37
Slanted/italic shape substitution implemented.	37
\zf@fontspec: Absorbed the comma into \zf@. .@options as to be more efficient when they are not defined.	36
Abstracted the long family name so the NFSS family is simple.	36
Incorporated \zf@get@feature@requests argument change.	36
Incorporated \zf@make@font@shapes change; removed \zf@options storage macro.	36
\zf@get@feature@requests: Absorbed comma into \zf@default@options, making \zf@current@options redundant.	38
Added an argument to eliminate the \zf@options macro.	38
Removed init stuff.	38
\zf@init: Taken from \zf@get@feature@requests.	38
\zf@make@feature: Now checks for OpenType feature.	39
\zf@make@font@shapes: \zf@scale@str eliminated.	37
Absorbed \IfEqFonts.	37
Added argument for \zf@get@feature@requests change.	37
Added code for SmallCaps . . . features.	37
Added logging of /B, /I, /BI failure.	37
Changed input syntax.	37
Incorporated \sidefault test into the \DeclareFontShape argument directly now that it's fully expanded.	37
Made local to hide \zf@fontname changes.	37
Removed \zf@scshape macro.	37
Removed \nfss@catcodes wrapper.	37
\zf@make@smallcaps: Now uses \zf@check@ot@feat.	39
\zf@partial@fontname: Implemented.	44
\zf@update@family: Now fully expands arguments.	38
\zf@update@ff: Removed ridiculous \zf@feature@separator code.	39
\zf@v@strnum: Implemented.	41
\zf@wordspace@parse: Implemented.	46

Index

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

Symbols	
\,	43
\@empty	103,
106, 173, 192, 207, 216, 228–230,	
263, 275, 278, 310–312, 316, 317,	
320–329, 336, 339, 356, 393, 401,	
406, 409, 415, 422, 424, 483, 494,	
509, 516, 538, 570, 598, 732, 737	
\@firstofone	319
\@font@info	1004
\@font@warning	1003, 1004
\@gobble	427
\@ifnextchar	87, 95
\@ifpackageloaded	1000, 1005, 1015–1017
\@nameuse	195
\@ne	437, 449, 461
\@nil	413–416, 427, 488, 499, 504, 514, 518
\@onlypreamble	79–82
\@tempa	90, 93, 98, 101, 107, 111, 262,
263, 265, 269, 294, 297–300, 303,	
313, 314, 339, 340, 356, 362, 392,	
393, 407, 409, 482, 483, 489, 493,	
494, 500, 505, 508, 509, 515, 520,	
522, 540, 542, 546, 569, 570, 592,	
594, 738, 980, 984, 987, 1031, 1033	
\@tempb	268,
269, 292, 293, 296, 298, 299, 395,	
398, 400–402, 406, 407, 541, 542,	
545, 546, 593, 594, 598, 618, 981, 982	
\@tempcnta	142, 153, 417, 419,
421, 423, 425, 433, 445, 457, 958, 965	
\@tempcntb	418–425, 430,
432, 435, 442, 444, 447, 453, 456, 459	
\@tempdima	
557, 559, 571–573, 575, 580, 585, 586	
\@tempdimb	558, 559, 572, 576, 581
\@tempdimc	559, 560, 573, 577, 582
\@tempfonta	264, 265
\@tempfontb	267, 268, 273
\@tempswafalse	225, 226, 431, 443, 455
\@tempswatruel	224, 434, 446, 458
\@zf@configfilefalse	29
\@zf@configfiletrue	28
\@zf@euencfalse	26
\@zf@euenctrue	27
\[.	42
\ \	19, 20, 37
\]	44
A	
\acute	1020
\addfontfeature	8, 115
\addfontfeatures	8, <u>104</u>
\advance	419, 421, 423, 425, 437, 449, 461
\aliasfontfeature	10–15, 26, <u>136</u>
\aliasfontfeatureoption	16–18, 26, <u>136</u>
\AtBeginDocument	<u>1000</u>
B	
\bar	1024
\begin	14, 23, 36
\begingroup	105, 162, 261, 555, 1030
\bfdefault	
209, 212, 232, 235, 239, 243, 1078,	
1081, 1084, 1087, 1088, 1090, 1091	
\bfseries	38
\breve	1025
C	
\c@zf@index	16, 431–433, 435, 437,
443–445, 447, 449, 455–457, 459, 461	
\c@zf@language	
.	18, 153, 332, 453, 457, 958, 965
\c@zf@newff	15, 118
\c@zf@script	17, 142, 330, 442, 445, 453, 457
\check	1026
\colon	1033, 1034
\cos	41
\csname	22, 23, 25, 109, 110, 120, 193
\cyrillicencoding	46, 49
D	
\ddot	1022
\DeclareFontFamily	203
\DeclareFontShape	295, 301
\DeclareMathAccent	1020–1029
\DeclareMathDelimiter	1066–1070
\DeclareMathSymbol	
.	1034, 1038–1041, 1043–1065, 1071
\DeclareOption	26–30

\DeclareRobustCommand	
..... 91, 99, 975, 988, 991, 994, 997	
\DeclareSymbolFont	1018, 1074
\DeclareTextFontCommand	978
\def	38, 42,
83–85, 88, 96, 102, 169, 175, 178,	
292, 331, 333, 414, 416, 466, 518,	
525, 528, 531, 534, 537, 568, 569, 618	
\def@cx	22, 118, 191
\defaultfontfeatures	7, 7, 46, 102
\define@choicekey	467
\define@key	8, 119, 139,
150, 378, 380, 464, 471, 476, 481,	
492, 503, 507, 524, 527, 530, 533,	
536, 539, 563, 583, 587, 590, 619,	
623, 626, 629, 632, 730, 735, 784, 955	
\Delta	1054
\document	48
\documentclass	1
\dot	1028
\dots	37, 38
E	
\edef ... 22, 90, 98, 107, 164, 262, 265,	
268, 272, 284, 294, 298, 300, 313,	
340, 346, 392, 395, 398, 400, 402,	
407, 468, 470, 472, 474, 477, 479,	
482, 485, 490, 493, 496, 501, 506,	
508, 511, 520, 522, 526, 529, 532,	
535, 540, 541, 545, 549, 553, 579,	
586, 592, 593, 634, 738, 980, 981, 984	
\egroup	1032
\else	41, 146, 157, 176,
188, 210, 219, 233, 237, 241, 271,	
278, 280, 283, 286, 351, 360, 370,	
396, 399, 408, 422, 424, 436, 448,	
460, 486, 497, 512, 521, 544, 548,	
574, 596, 603, 608, 611, 641, 642,	
787, 962, 969, 1008, 1037, 1042, 1086	
\em	1000
\emph	1000
\encodingdefault	43
\end	32, 39, 50
\endcsname 22, 23, 25, 109, 110,	
120, 123, 130, 186, 187, 193, 196, 983	
\endgroup	112, 247, 289, 562, 1036
\ExecuteOptions	31
\exp	43
\expandafter	22, 23, 25, 188,
189, 193, 422, 424, 427, 599, 601, 602	
F	
\f@encoding	983
\f@family	109, 110, 983
\f@series	983
\f@shape	982
\f@size .. 171, 181, 264, 267, 469, 473, 478	
\fi	44, 125,
132, 148, 160, 170, 179, 180, 182,	
189, 194, 213, 214, 222, 223, 225,	
226, 236, 240, 244–246, 256, 259,	
266, 280–282, 286–288, 304, 308,	
341, 342, 347, 348, 351, 352, 363,	
364, 375, 376, 403–405, 410, 411,	
422, 424, 438, 450, 462, 491, 502,	
517, 523, 537, 550, 551, 567, 578,	
607, 615–617, 636, 640, 646–648,	
734, 739, 790, 972, 973, 985, 986,	
1014, 1035, 1072, 1073, 1089, 1097	
\font 171, 181, 264, 267, 293, 469, 473,	
478, 557, 580–582, 586, 595, 602, 610	
\font@warning	1092
\fontdimen	557,
558, 571, 575–577, 580–582, 585, 586	
\fontfamily	53, 92, 100, 113
\fontname	147, 158,
265, 268, 359, 373, 605, 613, 645, 970	
\fontshape	977, 987
\fontspec	4, 20, 25, 34, 51
\frenchspacing	12
G	
\g@addto@macro	48, 595, 600, 610
\Gamma	1053
\gdef@cx	22, 199–201
\global	142, 153
\grave	1021
H	
\hat	1027
\hyphenchar	595, 601, 610
I	
\if	519
\if@tempw 141, 152, 227, 345, 367, 957, 964	
\if@zf@configfile	13, 1093
\if@zf@euenc	14, 37
\ifcase	250
\ifcsname ... 123, 130, 186, 187, 196, 983	
\ifdefined	1082
\ifnum	254, 394,
432, 433, 444, 445, 456, 457, 599, 609	
\ifodd	397

<code>\ifx</code> 168, 173, 207, 216, 228–230, 263, 269, 275, 278, 280, 286, 299, 339, 356, 393, 401, 406, 422, 424, 483, 494, 509, 542, 546, 570, 594, 598, 732, 737, 982, 1033	<code>\mathit</code> 989, 1013, 1077, 1085, 1088
<code>\ifzf@atsui</code> 6, 174, 337, 354	<code>\mathopen</code> 1066, 1068
<code>\ifzf@firsttime</code> 1, 306, 350, 537, 565, 643	<code>\mathord</code> 1070, 1071
<code>\ifzf@icu</code> 7, 177, 343, 351, 365, 633, 641, 785	<code>\mathpunct</code> 1034, 1040
<code>\ifzf@math@euler</code> 9, 1037	<code>\mathrel</code> 1039, 1065
<code>\ifzf@math@lucida</code> 10, 1042	<code>\mathring</code> 1029
<code>\ifzf@mm</code> 8, 637, 642	<code>\mathrm</code> 43, 1076, 1083, 1087
<code>\ifzf@nobf</code> 2, 206, 225	<code>\mathsf</code> 1079, 1090
<code>\ifzf@noit</code> 3, 215, 226	<code>\mathtt</code> 1080, 1091
<code>\ifzf@nosc</code> 4, 276	<code>\mddefault</code> 205, 218, 221, 1074–1077, 1079, 1080, 1083, 1085
<code>\ifzf@package@babel@loaded</code> 12	<code>\MessageBreak</code> 1010–1012
<code>\ifzf@package@euler@loaded</code> 11, 1006	<code>\multi@alias@key</code> 136, <u>384</u>
<code>\ifzf@tfm</code> 5	<code>\multiply</code> 419, 421, 423
<code>\ignorespaces</code> 54, 114	
<code>\indent</code> 20	N
<code>\infty</code> 43	<code>\newAATfeature</code> 24, <u>122</u>
<code>\InputIfFileExists</code> 1094	<code>\newcommand</code> 7, 19–21, 51, 55, 59, 63, 67, 70, 73, 76, 86, 94, 102, 104, 116, 122, 129, 136–138, 149, 161, 248, 260, 290, 305, 309, 315, 335, 349, 353, 377, 379, 381, 384, 391, 412, 426, 428, 440, 452, 554, 979
<code>\int</code> 43	<code>\newcount</code> 15–18
<code>\itdefault</code> 218, 221, 232, 235, 239, 243, 280, 286, 298, 302, 990, 996, 1077, 1085, 1088	<code>\newcounter</code> 189
<code>\itshape</code> 37, 38, <u>988</u>	<code>\newfeaturecode</code> 7
	<code>\newfontface</code> 5, <u>86</u>
K	<code>\newfontface@i</code> 95, 96
<code>\key@ifundefined</code> 126, 133, 385, 386	<code>\newfontfamily</code> 5, 5, <u>86</u>
<code>\keyval@alias@key</code> 137, <u>381</u> , 389, 390, 622	<code>\newfontfamily@i</code> 87, 88
	<code>\newfontfeature</code> 24, <u>116</u>
L	<code>\newfontinstance</code> 5
<code>\Lambda</code> 1056	<code>\newfontlanguage</code> 23, <u>149</u> , 827–954
<code>\Large</code> 24	<code>\newfontscript</code> 23, 23, 24, 25, <u>138</u> , 791–826
<code>\LaTeX</code> 19	<code>\newICUfeature</code> 24, <u>129</u>
<code>\latinencoding</code> 47, 50	<code>\newif</code> 1–14
<code>\let</code> 5, 25, 30, 43, 45–47, 49, 50, 57, 61, 65, 69, 72, 75, 78, 103, 106, 115, 165, 167, 273, 310–312, 316–329, 336, 409, 489, 500, 505, 515, 1003, 1004, 1032, 1092	<code>\next</code> 1032
<code>\let@cc</code> <u>22</u> , 382, 383	<code>\noexpand</code> 91, 92, 99, 100, 108, 294, 300, 313, 738
<code>\loop</code> 432, 444, 456	<code>\normalfont</code> 58, 62, 66
	<code>\not@math@alphabet</code> 976, 989, 992, 995, 998
M	<code>\numexpr</code> 400
<code>\mathalpha</code> 1020–1029, 1043–1063	
<code>\mathbf</code> 1013, 1078, 1084	O
<code>\mathbin</code> 1064	<code>\Omega</code> 1063
<code>\mathcal</code> 43	<code>\or</code> 252, 257
<code>\mathchardef</code> 1031	
<code>\mathclose</code> 1038, 1041, 1067, 1069	P
<code>\mathdollar</code> 1071	<code>\PackageError</code> 19
	<code>\PackageInfo</code> 21
	<code>\PackageWarning</code> 20
	<code>\pagestyle</code> 15

\Phi	1061	\textsf	17, 19, 41
\Pi	1058	\textsi	974
\pi	41	\the	118, 193, 580–582, 586
\pm	41	\Theta	1055
\ProcessOptions	32	\tilde	1023
\protect	1013	\ttdefault	65, 85
\providecommand	22–24, 974	\two@digits	738
\Psi	1062	\typeout	1095, 1096
R		U	
\ratio	559	\unless	123, 130, 168, 186, 196, 206, 215, 263, 276, 339, 350, 393, 397, 401, 406, 537, 565, 732, 737
\relax	142, 153, 168, 318, 330, 332, 400, 418, 519, 595, 610, 958, 965, 976, 992, 995, 998	\updefault	205, 209, 212, 999, 1074–1076, 1078–1081, 1083, 1084, 1087, 1090, 1091
\repeat	439, 451, 463	\upshape	988
\RequirePackage	33, 35, 36, 39, 40	\Upsilon	1060
\RequireXeTeX	34	\usepackage	3–5
\rmdefault	57, 83	\UTFencname	45
\rmfamily	556	V	
S		\verb	20, 34, 46
\scdefault ..	280, 286, 990, 993, 996, 999	X	
\scshape	37, 988	\xdef	23, 143–145, 154–156, 195, 307, 351, 560, 959–961, 966–968
\section	17	\XeTeX	19
\selectfont	53, 92, 100, 113, 977, 987	\XeTeXcharglyph	599, 609
\setboldmathrm	6, 67	\XeTeXcountvariations	254
\setkeys ..	166, 313, 378, 475, 480, 731, 736	\XeTeXfeaturename	392
\setlength ..	557–559, 571, 575–577, 585	\XeTeXfonttype	250
\SetMathAlphabet	1076–1080, 1083–1085, 1087, 1088, 1090, 1091	\XeTeXisexclusivefeature	394
\setmathrm	6, 67	\XeTeXOTcountfeatures	453
\setmathsf	6, 67	\XeTeXOTcountlanguages	442
\setmathtt	6, 67	\XeTeXOTcountscripts	430
\setmonofont	4, 10, 34, 55	\XeTeXOTfeaturetag	457
\setromanfont	4, 8, 34, 55	\XeTeXOTlanguagetag	445
\setsansfont	4, 9, 34, 55	\XeTeXOTscripttag	433
\SetSymbolFont	1019, 1075, 1081	\XeTeXselectorname	395, 398, 400
\sfdefault	61, 84	\Xi	1057
\sffamily	38	\XKV@rm	167, 732, 737
\sidefault		\XKV@tfam	358, 372
	280, 286, 974, 977, 990, 993, 996, 999	\XKV@tkey	358, 372
\Sigma	1059	Z	
\sishape	974	\z@	417, 431, 443, 455
\sldefault	301, 993	\zap@space	192, 516, 538
\slshape	991, 992	\zf@@	566, 568, 597, 618
\space	293, 561, 605, 613, 645	\zf@ii	566
\stepcounter	117, 188	\zf@iii	566
\string	293		
\strip@pt	560		
T			
\TeX	46		

\zf@adjust	272, 284, 292, 469, 473, 478, 520, 561
296, 302, 312, 579, 586, 595, 600, 610	
\zf@atsuifalse	52, 56,
249	60, 64, 68, 71, 74, 77, 89, 97, 108, <u>161</u>
\zf@atsuitrue	184, 291, <u>309</u>
253	
\zf@basefont	249
... 147, 158, 171, 181, 250, 254,	249
273, 359, 373, 392, 394, 395, 398,	
400, 430, 433, 442, 445, 453, 457,	
469, 473, 478, 558, 571, 575–577,	
585, 599, 605, 609, 613, 645, 970	
\zf@bf	163, <u>315</u>
207, 211, 229, 238, 321, 489	
\zf@bf@feat	216, 220, 230, 234, 322, 500
209, 212, 326, 528	
\zf@bfit	218, 221, 327, 531
228, 242, 323, 505	
\zf@bfit@feat 232, 235, 239, 243, 328, 534	
\zf@calc@scale	412, 429, 441
543, 547, <u>554</u>	
\zf@check@one@char	413, 414, 427
597, 618	
\zf@check@ot@feat	415, 416
344, 366, <u>452</u>	
\zf@check@ot@lang ... 151, <u>440</u> , 956, 963	
\zf@check@ot@script	154, 333, 374, 959, 966
140, <u>428</u>	
\zf@DeclareFontShape . 274, 279, 285, <u>290</u>	
\zf@default@options	
102, 103, 106, 198, 200, 313	
\zf@define@feature@option	
128, 135, <u>377</u> , 650–673, 675–683,	338, 355, <u>391</u>
685–692, 694–705, 707–709,	
711–716, 718–724, 726–729,	\zf@make@feature 8, <u>353</u> , 380, 733, 738, 786
741–751, 753–759, 761–768, 770–782	
\zf@define@font@feature 124,	\zf@make@font@shapes 204, 208,
131, <u>377</u> , 649, 674, 684, 693, 706,	211, 217, 220, 231, 234, 238, 242, <u>260</u>
710, 717, 725, 740, 752, 760, 769, 783	
\zf@enc	\zf@make@smallcaps
38, 40, 42, 43, 45–47,	277, <u>335</u>
49, 50, 203, 295, 301, 1074–1081,	\zf@math@eulertrue
1083–1085, 1087, 1088, 1090, 1091	1007
\zf@family	\zf@math@lucidatrue
53, 57,	1015–1017
61, 65, 69, 72, 75, 78, 92, 100, 113,	\zf@merge@shape . <u>979</u> , 990, 993, 996, 999
195, 196, 199–201, 203, 295, 301, 302	
\zf@family@long	\zf@mmfalse
144, 155, 165, 186, 191, 195, 307,	249
470, 474, 479, 485, 490, 496, 501,	\zf@mmtrue
506, 511, 526, 529, 532, 535, 960, 967	255
\zf@ff	\zf@nobffalse
202, 292, 310, 351	487
\zf@firsttimefalse	\zf@nobftrue
185	465, 484
\zf@firsttimetrue	\zf@noitfalse
183	498
\zf@font@feat ... 167, 184, 205, 209,	\zf@noittrue
212, 218, 221, 232, 235, 239, 243, 317	465, 495
\zf@font@path	\zf@noscfalse
168, 169, 318, 466	513
\zf@font@warning	\zf@nosctrue
1003, 1092	510
\zf@font@wrap	\zf@package@euler@loadedfalse .. 1001
169, 171, 181, 264, 267, 292, 319, 473	\zf@package@euler@loadedtrue ... 1000
\zf@fontname	\zf@PackageError 19, 387, 604, 612, 1009
164, 165, 171,	\zf@PackageInfo 21, 30, 197, 270, 293, 561
181, 199, 202, 204, 208, 217, 231,	\zf@PackageWarning
	20, 30,
	127, 134, 147, 158, 357, 371, 644, 970
	\zf@partial@fontname
	488, 499, 504, 514, <u>518</u>
	\zf@pre@ff 145, 156, 202, 292, 316, 961, 968
	\zf@rmboldmaths
	72, 1082–1085
	\zf@rmmaths
	69, 83, 1074–1078, 1081, 1087, 1088
	\zf@sc
	275, 284, 324, 515
	\zf@sc@feat
	280, 286, 329, 537
	\zf@scale 296, 311, 549, 552, 553, 560, 561
	\zf@script@name . 143, 159, 331, 374, 971
	\zf@set@font@type
	172, <u>248</u>
	\zf@sfmaths
	75, 84, 1079, 1090
	\zf@smallcaps ... 278, 279, 336, 340, 346

\zf@suffix	171, 173, 175, 178, 181, 202, 264, 267, 292, 320, 468, 469, 472, 473, 477, 478, 634
\zf@tfm	251
\zf@tfmfalse	249
\zf@ttmaths	78, 85, 1080, 1091
\zf@up@feat	205, 325, 525
\zf@update@family . . .	120, <u>305</u> , 361, 368, 516, 538, 552, 564, <u>584</u> , 588, 591, 620, 624, 627, 630, 635, 638, 788
\zf@update@ff	121, <u>349</u> , 362, 369, 589, 621, 625, 628, 631, 639, 789
\zf@v@strnum	<u>412</u> , 454
\zf@wordspace@parse	<u>566</u> , <u>568</u>