

# The fontspec package

Will Robertson

2006/08/20    v1.12

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>	6.6	Contextuals	14
1.1	Usage	2	6.7	Vertical position	15
1.2	Warning	3	6.8	Fractions	16
1.3	About this manual	3	6.9	Variants	16
<b>2</b>	<b>Brief overview</b>	<b>3</b>	6.10	AAT Alternates	17
<b>3</b>	<b>Font selection</b>	<b>3</b>	6.11	Style	17
3.1	Default font families	4	6.12	Diacritics	18
3.2	Font instances for efficiency	4	6.13	Kerning	18
3.3	Arbitrary bold/italic/small caps fonts	5	6.14	CJK shape	19
3.4	Math(s) fonts	5	6.15	Character width	19
3.5	Miscellaneous font selecting details	6	6.16	Annotation	20
<b>4</b>	<b>Selecting font features</b>	<b>6</b>	6.17	Vertical typesetting	20
4.1	Default settings	6	6.18	AAT & Multiple Master font axes	21
4.2	Changing the currently selected features	7	6.19	OpenType scripts and languages	21
4.3	Priority of feature selection	7	<b>7</b>	<b>Defining new features</b>	<b>23</b>
4.4	Different features for different font shapes	7	7.1	Renaming existing features & options	25
<b>5</b>	<b>Font independent options</b>	<b>8</b>	<b>I</b>	<b>fontspec.sty</b>	<b>26</b>
5.1	Scale	8	<b>8</b>	<b>Implementation</b>	<b>26</b>
5.2	Mapping	9	8.1	Bits and pieces	26
5.3	Colour	9	8.2	Packages	26
5.4	Interword space	9	8.3	Encodings	27
5.5	Post-punctuation space	10	8.4	User commands	27
5.6	Letter spacing	10	8.5	Internal macros	31
5.7	The hyphenation character	10	8.6	keyval definitions	40
<b>6</b>	<b>Font-dependent features</b>	<b>11</b>	8.7	Italic small caps	53
6.1	Different font technologies: AAT and ICU	11	8.8	Selecting maths fonts	54
6.2	Optical font sizes	12	8.9	Option processing	57
6.3	Ligatures	13	<b>II</b>	<b>fontspec.cfg</b>	<b>58</b>
6.4	Letters	13	<b>III</b>	<b>fontspec-example.tex</b>	<b>58</b>
6.5	Numbers	14			

# 1 Introduction

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

However, it was still necessary to write cumbersome font definition files for L<sup>A</sup>TeX, since the NFSS had a lot more going on behind the scenes to allow easy commands like `\emph` or `\bfseries`.

This package almost entirely eliminates this need by providing a completely automatic way to select font families in L<sup>A</sup>TeX for arbitrary fonts. Furthermore, it allows (again, almost) total control over the selection of rich 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 L<sup>A</sup>TeX'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 L<sup>A</sup>TeX 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 will all not work correctly, as far as I can tell. Cyrillic and Latin-based languages, however, might—fontspec ensures at least that fonts should load correctly, but hyphenation and other matters aren't guaranteed.

### 1.1.1 Configuration

If you wish to customise any part of the fontspec interface (see later in this manual, Section 7 on page 23 and Section 7.1), this should be done by creating your own `fontspec.cfg` file,<sup>2</sup> which will be automatically loaded if it is found by XeTeX. Either place it in the same folder as the main document for isolated cases, or in a location that XeTeX searches by default, e.g., `~/Library/texmf/xelatex/`. The package option `[noconfig]` will suppress this behaviour under all circumstances.

### 1.1.2 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 tran-

---

<sup>1</sup><http://scripts.sil.org/xetex>

<sup>2</sup>An example is distributed with the package.

script (.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<sub>Y</sub>TeX using a variety of fonts to display various features that the package supports. You will not be able to typeset the documentation if you do not have all of these fonts, although I've used as many Mac OS X pre-installed fonts as possible. Running normal L<sup>A</sup>T<sub>E</sub>X (*i.e.*, without X<sub>Y</sub>TeX) on this file will generate the `fontspec.sty` file if this is required for some odd reason.

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

## 2 Brief overview

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

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

## 3 Font selection

`\fontspec` `\fontspec[<font features>]{<font name>}` is the base command of the package, used for selecting the specified *<font name>* in a L<sup>A</sup>T<sub>E</sub>X family. The font features argument accepts comma separated *<font feature>*=*<option>* lists; these will not be fully described until Section 6 on page 11.

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

<p>The five boxing wizards jump quickly.  <i>The five boxing wizards jump quickly.</i>  <b>THE FIVE BOXING WIZARDS JUMP QUICKLY.</b>  <i>THE FIVE BOXING WIZARDS JUMP QUICKLY.</i>  <b>The five boxing wizards jump quickly.</b>  <i>The five boxing wizards jump quickly.</i>  <b>THE FIVE BOXING WIZARDS JUMP QUICKLY.</b>  <i>THE FIVE BOXING WIZARDS JUMP QUICKLY.</i></p>	<pre>\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 }</pre>
--	--

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<sub>Y</sub>TeX in any way whatsoever: every font that is installed in the operating system may be accessed.

### 3.1 Default font families

`\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:

<p>Pack my box with five dozen liquor jugs.  <i>Pack my box with five dozen liquor jugs.</i>  <b>Pack my box with five dozen liquor jugs.</b></p>	<pre>\setromanfont{Baskerville} \setsansfont[Scale=0.86]{Skia} \setmonofont[Scale=0.8]{Monaco} \rmfamily\pangram\par \sffamily\pangram\par \ttfamily\pangram</pre>
---	--

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 8, 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:

<p>This font is used for <i>notes</i>.</p>	<pre>\newfontfamily\notefont{Didot} \notefont This font is used for \emph{notes}.</pre>
--	---

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}]
{Hoefler Text Italic}
\fancy where is all the vegemite

```

*where is all the vegemite*

This example is repeated in Section 6.6 on page 14.

### 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 may not even have bold or italic shapes, in which case a skilled (or lucky) designer may be able to choose well-matching accompanying shapes from a different font altogether, others can have a range of bold and italic fonts to choose 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 \\\
{\itshape Helvetica Neue UltraLight Italic} \\\
{\bfseries Helvetica Neue } \\\
{\bfseries\itshape Helvetica Neue Italic} \\\

```

Helvetica Neue UltraLight  
*Helvetica Neue UltraLight Italic*  
**Helvetica Neue**  
*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’:

```

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

```

Baskerville *Italic* **SemiBold Italic**

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

```

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

```

Roman 123  
SMALL CAPS 456

### 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<sup>A</sup>T<sub>E</sub>X freezes the maths fonts after this stage of the processing. The `fontspec` package must also be loaded after any maths font packages (*e.g.*, `euler`) to be successful. (Actually, it is *only* `euler` that is the problem.)

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

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

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

and this would allow you to typeset something like this:

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

### 3.5 Miscellaneous font selecting details

By the way, from v1.9, `\fontspec` and `\addfontfeatures` will now ignore follow-  
ing 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 L<sup>A</sup>T<sub>E</sub>X. The font features and their options are described  
in Section 6 on page 11, 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 selec-  
tion command: a default feature set, so to speak. This may be defined with the  
`\defaultfontfeatures{font features}` command. New calls of `\defaultfont-`  
`features` 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{<font features>}` command allows font features to be changed without knowing what features are currently selected or even what font is being used. A good example of this could be to add a hook to all tabular material to use monospaced numbers, as shown in the following example:

<p>‘In 1842, 999 people sailed 97 miles in 13 boats. In 1923, 111 people sailed 54 miles in 56 boats.’</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 &amp; People &amp; Miles &amp; Boats \\\ \midrule 1842 &amp; 999 &amp; 75 &amp; 13 \\\ 1923 &amp; 111 &amp; 54 &amp; 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  $X_{\text{TeX}}$ , 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 ( $\leftarrow$ ) 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<sub>Y</sub>TeX 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 6 and Section 6 on page 11, uses X<sub>Y</sub>TeX 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<sub>E</sub>X 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<sub>Y</sub>TeX, 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<sub>Y</sub>TeX 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<sub>E</sub>X, 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,  $\text{\TeX}$  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 13).

## 5.7 The hyphenation character

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

Below, Adobe Garamond Pro’s uppercase hyphenation character<sup>3</sup> 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 13).

The xltextra package redefines L<sup>A</sup>T<sub>E</sub>X’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 1<sup>st</sup> 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<sub>Y</sub>L<sup>A</sup>T<sub>E</sub>X 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 1.1.2 on page 2 these warnings can be suppressed by selecting the [quiet] package option.

### 6.1 Different font technologies: AAT and ICU

X<sub>Y</sub>L<sup>A</sup>T<sub>E</sub>X 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.

<sup>3</sup>I 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 21 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 X<sub>Y</sub>TeX *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 21 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<sup>A</sup>T<sub>E</sub>X, 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	<code>\fontspec[Ligatures=Rare]{Hoefler Text}</code>
strict firefly	strict firefly
	<code>\fontspec[Ligatures=NoCommon]{Hoefler Text}</code>
	strict firefly
Rare: Ð Þ ð þ	<code>\fontspec</code>
Logos: 🍏	<code>[Ligatures={Rare,Logos,Rebus,Diphthong}]</code>
Rebus: %	<code>{Palatino}</code>
Diphthong: Æ Œ æ œ	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<sub>E</sub>X trick of splitting up a ligature with an empty brace pair does not work in X<sub>Y</sub>T<sub>E</sub>X; you must use a 0 pt kern or \hbox (*e.g.*, \null) to split the characters up.

### 6.4 Letters

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

The Letters 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	<code>\fontspec[Letters=Uppercase]{Palatino}</code>
this sentence no verb	THIS Sentence no verb
This Sentence No Verb	<code>\fontspec[Letters=Lowercase]{Palatino}</code>
	THIS Sentence no verb
	<code>\fontspec[Letters=InitialCaps]{Palatino}</code>
	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.<sup>4</sup> 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 18). 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 7.

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.)

<sup>4</sup>If you want automatic uppercase letters, look into the \MakeUppercase command, as defined by L<sup>A</sup>T<sub>E</sub>X.



*where is all the vegemite*

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

'Inner' fwashes can *fometimes*  
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 <sup>superior</sup> <sub>inferior</sub>  
1<sup>st</sup> 2<sup>nd</sup> 3<sup>rd</sup> 4<sup>th</sup> 0<sup>th</sup> 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

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

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

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

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

```
\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 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 23 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]{Hoefler Text Italic}</code>
<i>Sphinx Of Black Quartz, JUDGE Mr</i>	<code>Sphinx Of Black Quartz, {\scshape Judge My Vow} \</code>
<i>Vow</i>	<code>\fontspec[Alternate=1]{Hoefler Text Italic}</code>
	<code>Sphinx Of Black Quartz, {\scshape Judge My Vow}</code>

See Section 7 on page 23 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`,<sup>5</sup> `TallCaps`, or `TitlingCaps`.

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

ICU supported options are `Alternate`, `Italic`, `Historic`, `Ruby`,<sup>5</sup> `Swash`, `TitlingCaps`, `HorizontalKana`, and `VerticalKana`.

<i>K Q R k v w y</i>	<code>\fontspec{Warnock Pro}</code>
<i>K Q R k v w 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>M Q Z</i>	<code>\fontspec{Adobe Jenson Pro}</code>
<i>M Q Z</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>

<sup>5</sup>‘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<sub>E</sub>X 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 13, 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.<sup>6</sup> (←) 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]{ワカヨタレン}%
		abcdef	\makebox[2.5cm][l]{abcdef}}
ようこそ	ワカヨタレン	abcdef	\fontspec{Hiragino Mincho Pro}
ようこそ	ワカヨタレン	a b c d e f	{\addfontfeature{CharacterWidth=Proportional}\test}\
ようこそ	ワカヨタレン	abcdef	{\addfontfeature{CharacterWidth=Full}\test}\
			{\addfontfeature{CharacterWidth=Half}\test}

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

<sup>6</sup>Apple seems to be adapting its AAT features in this regard (at least in the fonts it distributes with Mac OS X) to have a one-to-one correspondence with the equivalent OpenType features. Previously AAT was more fine grained, but naturally they’re not documenting their AAT tables any more, so if the following features don’t work for a specific font let me know and I’ll try and see if anything can be salvaged from the situation.

— 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  $90^\circ$ , 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<sub>La</sub>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<sub>La</sub>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 12.

## 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 23 and Table 2 on page 24, 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 11.

### 6.19.1 Script examples

In the following examples, the same font is used to typeset the verbatim input and the Xe<sub>La</sub>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 số mỗ	<code>\fontspec{Doulos SIL}</code>
cáp số mỗ	<code>cáp số mỗ \</code>
cáp số mỗ	<code>\addfontfeature{Language=Vietnamese}</code>
	<code>cáp số mỗ</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>
Ş ş Ţ ţ	<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 following 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<sub>Y</sub>TeX 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 21) and the Local forms 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<sub>Y</sub>TeX 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	Garó	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	Hawaiian	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{<font feature>}{<existing name>}{<new name>}`:

Scientific  
 Inferior: 12345

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

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

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

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

## File I

# fontspec.sty

## 8 Implementation

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

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

Only proceed if it is  $\text{\LaTeX}$  that is doing the typesetting.

```
1 \RequirePackage{ifxetex}
2 \RequireXeTeX
```

### 8.1 Bits and pieces

Counters, conditionals, ...

```
3 \newif\ifzf@firsttime
4 \newif\ifzf@nobf
5 \newif\ifzf@noit
6 \newif\ifzf@nosc
7 \newif\ifzf@tfm
8 \newif\ifzf@atsui
9 \newif\ifzf@icu
10 \newif\ifzf@mm
11 \newif\ifzf@math@euler
12 \newif\ifzf@math@lucida
13 \newif\ifzf@euler@package@loaded
14 \newif\ifzf@package@babel@loaded
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}}
```

### 8.2 Packages

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

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

### 8.3 Encodings

Frank Mittelbach has recommended using the ‘EUx’ family of font encodings to experiment with unicode. But we’re not using it until some things have been sorted out. (How xunicode should behave; being able to load the Latin Modern fonts by default.)

```
24 \def\zf@enc{U}
25 \let\encodingdefault\zf@enc
```

We’ll use the following when the time comes:

```
\def\zf@enc{EU1}
\RequirePackage[\zf@enc]{fontenc}
```

Dealing with a couple of the problems introduced by babel:

```
26 \let\cyrillicencoding\zf@enc
27 \let\latinencoding\zf@enc
28 \g@addto@macro\document{%
29   \let\cyrillicencoding\zf@enc
30   \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.

### 8.4 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 8.5 on page 31.

#### 8.4.1 Font selection

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

```
31 \newcommand*\fontspec[2][]{%
32   \zf@fontspec{#1}{#2}%
33   \fontfamily\zf@family\selectfont
34   \ignorespaces}
```

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

`\setsansfont`

`\setmonofont`

```
35 \newcommand*\setromanfont[2][]{%
```

```

36 \zf@fontspec{#1}{#2}%
37 \let\rmdefault\zf@family
38 \normalfont}
39 \newcommand*\setsansfont[2][]{%
40 \zf@fontspec{#1}{#2}%
41 \let\sfdefault\zf@family
42 \normalfont}
43 \newcommand*\setmonofont[2][]{%
44 \zf@fontspec{#1}{#2}%
45 \let\ttdefault\zf@family
46 \normalfont}

```

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

```

47 \newcommand*\setmathrm[2][]{%
48 \zf@fontspec{#1}{#2}%
49 \let\zf@rmmaths\zf@family}
50 \newcommand*\setboldmathrm[2][]{%
51 \zf@fontspec{#1}{#2}%
52 \let\zf@rmboldmaths\zf@family}
53 \newcommand*\setmathsf[2][]{%
54 \zf@fontspec{#1}{#2}%
55 \let\zf@sfmaths\zf@family}
56 \newcommand*\setmathtt[2][]{%
57 \zf@fontspec{#1}{#2}%
58 \let\zf@ttmaths\zf@family}
59 \@onlypreamble\setmathrm
60 \@onlypreamble\setboldmathrm
61 \@onlypreamble\setmathsf
62 \@onlypreamble\setmathtt

```

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

```

63 \def\zf@rmmaths{\rmdefault}
64 \def\zf@sfmaths{\sfdefault}
65 \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`). This command is used when a specific font instance needs to be referred to repeatedly (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.

```

66 \newcommand*\newfontfamily[1]{%
67 \ifnextchar[{\newfontfamily@i#1}{\newfontfamily@i#1[]}}
68 \def\newfontfamily@i#1[#2]#3{%
69 \zf@fontspec{#2}{#3}%

```

```

70 \edef\@tempa{%
71   \noexpand\DeclareRobustCommand\noexpand#1
72   {\noexpand\fontfamily{\zf@family}\noexpand\selectfont}}%
73 \@tempa}

\newfontface uses an undocumented feature of the BoldFont feature; if its argu-
ment is empty (e.g., BoldFont={}, then no bold font is searched for.

74 \newcommand*\newfontface[1]{%
75   \@ifnextchar[{\newfontface@i#1}{\newfontface@i#1[]}}
76 \def\newfontface@i#1[#2]#3{%
77   \zf@fontspec{BoldFont={},ItalicFont={},SmallCapsFont={},#2}{#3}%
78   \edef\@tempa{%
79     \noexpand\DeclareRobustCommand\noexpand#1
80     {\noexpand\fontfamily{\zf@family}\noexpand\selectfont}}%
81   \@tempa}

```

#### 8.4.2 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.

```

82 \newcommand*\defaultfontfeatures[1]{\def\zf@default@options{#1,}}
83 \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 they're saved and restored with `\zf@default@options@old`), so this means that the only added features to the font are strictly those specified by this command.

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

```

84 \newcommand*\addfontfeatures[1]{%
85   \let\zf@default@options@old\zf@default@options
86   \let\zf@default@options\@empty
87   \edef\zf@thisinfo{%
88     \edef\@tempa{%
89       \noexpand\zf@fontspec
90       {\csname zf@family@options\f@family\endcsname,#1}%
91       {\csname zf@family@fontname\f@family\endcsname}}%
92     \@tempa
93     \fontfamily\zf@family\selectfont
94     \let\zf@default@options\zf@default@options@old
95     \ignorespaces}
96 \let\addfontfeature\addfontfeatures

```

### 8.4.3 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.

```

97 \newcommand*\newfontfeature[2]{%
98   \stepcounter{zf@newff}%
99   \def@cx{+zf-#1}{+zf-\the\c@zf@newff}%
100  \define@key{zf}[options]{#1}[]{}%
101   \zf@update@family{\csname+zf-#1\endcsname}%
102   \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.

```

103 \newcommand*\newAATfeature[4]{%
104   \unless\ifcsname zf@options@#1\endcsname
105     \zf@define@font@feature{#1}%
106   \fi
107   \key@ifundefined{zf}[#1]{#2}{}{}%
108   \zf@PackageWarning{Option '#2' of font feature '#1' overwritten.}}%
109   \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.

```

110 \newcommand*\newICUfeature[3]{%
111   \unless\ifcsname zf@options@#1\endcsname
112     \zf@define@font@feature{#1}%
113   \fi
114   \key@ifundefined{zf}[#1]{#2}{}{}%
115   \zf@PackageWarning{Option '#2' of font feature '#1' overwritten.}}%
116   \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.

```

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

```

`\newfontscript`

```

119 \newcommand*\newfontscript[2]{%
120   \define@key{zf@feat}[Script]{#1}[]{}%
121   \zf@check@ot@script{#2}%
122   \if@tempwa
123     \global\c@zf@script\@tempcnta\relax
124     \xdef\zf@script@name{#1}%
125     \xdef\zf@family@long{\zf@family@long+script=#1}%

```

```

126     \xdef\zf@pre@ff{script=#2,\zf@pre@ff}%
127     \else
128         \zf@PackageWarning{Font \fontname\zf@basefont does not con-
tain script '#1'}%
129     \fi}}

```

\newfontlanguage

```

130 \newcommand*\newfontlanguage[2]{%
131     \define@key{zf@feat}{Lang}{#1}[]{}%
132     \zf@check@ot@lang{#2}%
133     \if@tempswa
134         \global\c@zf@language\@tempcnta\relax
135         \xdef\zf@language@name{#1}%
136         \xdef\zf@family@long{\zf@family@long+lang=#1}%
137         \xdef\zf@pre@ff{\zf@pre@ff language=#2,}%
138     \else
139         \zf@PackageWarning{Font \fontname\zf@basefont does not contain
140                             language '#1' for script '\zf@script@name'}%
141     \fi}}

```

## 8.5 Internal macros

\def@cx Natural counterparts to \@namedef.

```

\gdef@cx 142 \providecommand\def@cx[2]{\expandafter\edef\csname#1\endcsname{#2}}
143 \providecommand\gdef@cx[2]{\expandafter\xdef\csname#1\endcsname{#2}}
144 \providecommand\let@cc[2]{\expandafter\let\csname#1\expandafter\endcsname\csname#2\endcsname}

```

\zf@fontspec This is the command that defines font families for use. Given a list of font features 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.

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

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

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.

Finally, 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 worthless. (E.g., \zf@init and \zf@set@font@type.)

```

145 \newcommand*\zf@fontspec[2]{%
146     \begingroup
147     \zf@init

```



```

148 \edef\zf@fontname{#2}%
149 \font\zf@basefont="\zf@fontname" at \f@size pt
150 \let\zf@family@long\zf@fontname
151 \setkeys*[zf]{preparse}{#1}%
152 \edef\zf@font@feat{\zf@font@feat\XKV@rm}%

```

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

```

153 \zf@set@font@type
154 \ifx\zf@suffix\@empty
155   \ifzf@atsui
156     \def\zf@suffix{/AAT}%
157   \else
158     \ifzf@icu
159       \def\zf@suffix{/ICU}%
160     \fi
161   \fi
162   \font\zf@basefont="\zf@fontname\zf@suffix" at \f@size pt
163 \fi
164 \zf@firsttimetrue
165 \zf@get@feature@requests{\zf@font@feat}%
166 \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.

```

167 \unless\ifcsname zf@UID@\zf@family@long\endcsname
168   \ifcsname c@zf@famc@#2\endcsname
169     \expandafter\stepcounter\else
170     \expandafter\newcounter\fi
171     {zf@famc@#2}%
172   \def@cx{zf@UID@\zf@family@long}{%
173     \zap@space#2 \@empty
174     (\expandafter\the\csname c@zf@famc@#2\endcsname)}%
175   \fi
176   \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`.

```

177 \unless\ifcsname zf@family@fontname\zf@family\endcsname
178   \zf@PackageInfo{Defining font family for "#2"
179     with options [\zf@default@options #1]}%
180   \gdef@cx{zf@family@fontname\zf@family}{\zf@fontname}%
181   \gdef@cx{zf@family@options\zf@family}{\zf@default@options #1}%
182   \gdef@cx{zf@family@fontdef\zf@family}{\zf@fontname\zf@suffix:\zf@pre@ff\zf@ff}%

```

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

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

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

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

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

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

```
185 \unless\ifzf@nobf
186 \ifx\zf@bf\empty
187 \zf@make@font@shapes[\zf@fontname]{/B}{\bfdefault}{\updefault}{\zf@font@feat\zf@bf@feat}%
188 \else
189 \zf@make@font@shapes{\zf@bf}{\bfdefault}{\updefault}{\zf@font@feat\zf@bf@feat}%
190 \fi
191 \fi
```

And italic in the same way:

```
192 \unless\ifzf@noit
193 \ifx\zf@it\empty
194 \zf@make@font@shapes[\zf@fontname]{/I}{\mddefault}{\itdefault}{\zf@font@feat\zf@it@feat}%
195 \else
196 \zf@make@font@shapes{\zf@it}{\mddefault}{\itdefault}{\zf@font@feat\zf@it@feat}%
197 \fi
198 \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.

```
199 \@tempswatrue
200 \ifzf@nobf\@tempswafalse\fi
201 \ifzf@noit\@tempswafalse\fi
202 \if@tempswa
203 \ifx\zf@bfit\empty
204 \ifx\zf@bf\empty
205 \ifx\zf@it\empty
206 \zf@make@font@shapes[\zf@fontname]{/BI}{\bfdefault}{\itdefault}{\zf@font@feat\zf@bfit@feat}%
207 \else
208 \zf@make@font@shapes[\zf@it]{/B}{\bfdefault}{\itdefault}{\zf@font@feat\zf@bfit@feat}%
209 \fi
210 \else
211 \zf@make@font@shapes[\zf@bf]{/I}{\bfdefault}{\itdefault}{\zf@font@feat\zf@bfit@feat}%
```

```

212     \fi
213     \else
214     \zf@make@font@shapes{\zf@bfit}{\bfdefault}{\itdefault}{\zf@font@feat\zf@bfit@feat}%
215     \fi
216     \fi
217     \fi
218 \endgroup
219 }

```

### 8.5.1 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.

```

220 \newcommand*\zf@set@font@type{%
221   \zf@tfmfalse \zf@atsuifalse \zf@icufalse \zf@mmfalse
222   \ifcase\XeTeXfonttype\zf@basefont
223     \zf@tfm
224   \or
225     \zf@atsui true
226     \ifnum\XeTeXcountvariations\zf@basefont > 0
227       \zf@mm true
228     \fi
229   \or
230     \zf@icutrue
231   \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<sub>Y</sub>TEX'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.

```

232 \newcommand*\zf@make@font@shapes[5][]{%
233   \bgroup
234   \edef\@tempa{#1}%
235   \ifx\@tempa\empty\else
236     \font\@tempfonta="#1\zf@suffix" at \f@size pt
237     \edef\@tempa{\fontname\@tempfonta}%
238   \fi
239   \font\@tempfontb="#1#2\zf@suffix" at \f@size pt
240   \edef\@tempb{\fontname\@tempfontb}%

```

```

241 \ifx\@tempa\@tempb
242 \zf@PackageInfo{Could not resolve font #1#2 (it might not exist)}%
243 \else
244 \edef\zf@fontname{#1#2}%
245 \let\zf@basefont\@tempfontb
246 \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 8.7 on page 53 for the code that enables this usage.

```

247 \ifx\zf@sc\@empty
248 \unless\ifzf@nosc
249 \zf@make@smallcaps
250 \ifx\zf@smallcaps\@empty\else
251 \zf@DeclareFontShape[\zf@smallcaps]{#3}
252 {\ifx#4\itdefault\sidefault\else\scdefault\fi}{#5\zf@sc@feat}%
253 \fi
254 \fi
255 \else
256 \edef\zf@fontname{\zf@sc}%
257 \zf@DeclareFontShape{#3}
258 {\ifx#4\itdefault\sidefault\else\scdefault\fi}{#5\zf@sc@feat}%
259 \fi
260 \fi
261 \egroup}

```

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.

```

262 \newcommand\zf@DeclareFontShape[4]{}{%
263 \zf@get@feature@requests{#4}%
264 \def\@tempb{"\zf@fontname\zf@suffix:\zf@pre@ff\zf@ff#1"}%
265 \zf@PackageInfo{\string\font\space is \@tempb}%
266 \edef\@tempa{\noexpand
267 \DeclareFontShape{\zf@enc}{\zf@family}{#2}{#3}
268 {<->\zf@scale\@tempb}{\zf@adjust}}%
269 \@tempa
270 \edef\@tempa{#3}\edef\@tempb{\itdefault}%
271 \ifx\@tempa\@tempb
272 \edef\@tempa{\noexpand
273 \DeclareFontShape{\zf@enc}{\zf@family}{#2}{\sldefault}
274 {<->sub*\zf@family/#2/\itdefault}{\zf@adjust}}%
275 \@tempa
276 \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.

```

277 \newcommand*\zf@update@family}[1]{%
278   \ifzf@firsttime
279     \xdef\zf@family@long{\zf@family@long#1}%
280   \fi}

```

### 8.5.2 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.

```

281 \newcommand*\zf@get@feature@requests[1]{%
282   \let\zf@ff      \@empty
283   \let\zf@scale   \@empty
284   \let\zf@adjust  \@empty
285   \edef\@tempa{\noexpand\setkeys[zf]{options}\zf@default@options#1}%
286   \@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.

```

287 \newcommand*\zf@init{%
288   \let\zf@pre@ff      \@empty
289   \let\zf@font@feat   \@empty
290   \let\zf@suffix      \@empty
291   \let\zf@bf          \@empty
292   \let\zf@it          \@empty
293   \let\zf@bfit        \@empty
294   \let\zf@sc          \@empty
295   \let\zf@up@feat     \@empty
296   \let\zf@bf@feat     \@empty
297   \let\zf@it@feat     \@empty
298   \let\zf@bfit@feat   \@empty
299   \let\zf@sc@feat     \@empty
300   \c@zf@script 1818326126\relax
301   \def\zf@script@name{Latin}%
302   \c@zf@language 0\relax
303   \def\zf@language@name{Default}%
304 }

```

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

```

305 \newcommand*\zf@make@smallcaps{%
306   \let\zf@smallcaps\@empty
307   \ifzf@atsui
308     \zf@make@aat@feature@string{3}{3}%
309     \unless\ifx\@tempa\@empty
310       \edef\zf@smallcaps{\@tempa;}%
311   \fi

```

```

312 \fi
313 \ifzf@icu
314 \zf@check@ot@feat{+smcp}%
315 \if@tempswa
316 \edef\zf@smallcaps{+smcp,}%
317 \fi
318 \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.

```

319 \newcommand*\zf@update@ff[1]{%
320 \unless\ifzf@firsttime
321 \xdef\zf@ff{\zf@ff #1\ifzf@icu,\else;\fi}%
322 \fi}

```

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

```

323 \newcommand*\zf@make@feature[3]{%
324 \ifzf@atsui
325 \zf@make@aat@feature@string{#1}{#2}%
326 \ifx\@tempa\@empty
327 \zf@PackageWarning{%
328 AAT feature '\XKV@tfam=\XKV@tkey'
329 (#1,#2) not available in font \fontname\zf@basefont}%
330 \else
331 \zf@update@family{+#1,#2}%
332 \zf@update@ff\@tempa
333 \fi
334 \fi
335 \ifzf@icu
336 \zf@check@ot@feat{#3}%
337 \if@tempswa
338 \zf@update@family{#3}%
339 \zf@update@ff{#3}%
340 \else
341 \zf@PackageWarning{%
342 OpenType feature '\XKV@tfam=\XKV@tkey' (#3)
343 not available in font \fontname\zf@basefont, script
344 '\zf@script@name', language '\zf@language@name'}%
345 \fi
346 \fi}

```

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

```

\zf@define@feature@option 347 \newcommand*\zf@define@font@feature[1]{%
348 \define@key[zf]{options}{#1}{\setkeys[zf@feat]{#1}{##1}}}
349 \newcommand*\zf@define@feature@option[5]{%
350 \define@key[zf@feat]{#1}{#2}[]{\zf@make@feature{#3}{#4}{#5}}}

```

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

```

351 \newcommand*\keyval@alias@key[4][KV]{%
352   \let@cc{#1@#2@#4}{#1@#2@#3}%
353   \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.

```

354 \newcommand*\multi@alias@key[2]{
355   \key@ifundefined[zf]{preparse}{#1}
356   {\key@ifundefined[zf]{options}{#1}
357     {\zf@PackageError{The feature #1 doesn't appear to be defined}
358       {It looks like you're trying to rename a feature that doesn't exist.}}
359     {\keyval@alias@key[zf]{options}{#1}{#2}}}
360   {\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 8.5.2).

```

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

```

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

```

364     \ifnum\XeTeXisexclusivefeature\zf@basefont #1 > 0
365     \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:

```

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

```

If the selector is *odd*, it corresponds to switching the feature off. But  $\text{\XeTeX}$  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.

```

369     \else
370     \edef\@tempb{\XeTeXselectorname\zf@basefont #1 \numexpr#2-1\relax}%
371     \unless\ifx\@tempb\@empty
372     \edef\@tempb{!\@tempb}%
373     \fi
374     \fi

```

Finally, save out the feature string in `\@tempa`, which will remain empty if the feature doesn't exist.

```

375     \fi
376     \unless\ifx\@tempb\@empty
377     \edef\@tempa{\@tempa=\@tempb}%
378     \fi
379     \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.

```

380 \newcommand\zf@iv@strnum[1]{%
381   \zf@iv@strnum@i#1 \@nil}
382 \def\zf@iv@strnum@i#1 \@nil{%
383   \zf@iv@strnum@ii#1\@empty\@empty\@nil}
384 \def\zf@iv@strnum@ii#1#2#3#4#5\@nil{%
385   \@tempcnta\z@
386   \@tempcntb`#1\relax
387   \multiply\@tempcntb"1000000\advance\@tempcnta\@tempcntb
388   \@tempcntb`#2
389   \multiply\@tempcntb"10000\advance\@tempcnta\@tempcntb
390   \expandafter\@tempcntb\ifx\@empty#332\else`#3\fi
391   \multiply\@tempcntb"100\advance\@tempcnta\@tempcntb
392   \expandafter\@tempcntb\ifx\@empty#432\else`#4\fi
393   \advance\@tempcnta\@tempcntb}
394 \newcommand\zf@v@strnum[1]{%
395   \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.

```

396 \newcommand\zf@check@ot@script[1]{%
397   \zf@iv@strnum{#1}%
398   \@tempcntb\XeTeXOTcountscript\zf@basefont
399   \c@zf@index\z@ \@tempswafalse
400   \loop\ifnum\c@zf@index<\@tempcntb
401     \ifnum\XeTeXOTscripttag\zf@basefont\c@zf@index=\@tempcnta
402       \@tempswatrue
403       \c@zf@index\@tempcntb
404     \else
405       \advance\c@zf@index\@ne
406     \fi
407   \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’.

```

408 \newcommand\zf@check@ot@lang[1]{%
409   \zf@iv@strnum{#1}%
410   \@tempcntb\XeTeXOTcountlanguages\zf@basefont\c@zf@script
411   \c@zf@index\z@ \@tempswafalse
412   \loop\ifnum\c@zf@index<\@tempcntb
413     \ifnum\XeTeXOTlanguage\zf@basefont\c@zf@script\c@zf@index=\@tempcnta
414       \@tempswatrue
415       \c@zf@index\@tempcntb
416     \else
417       \advance\c@zf@index\@ne
418     \fi
419   \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’.

```

420 \newcommand*\zf@check@ot@feat[1]{%
421   \@tempcntb\XeTeXOTcountfeatures\zf@basefont\c@zf@script\c@zf@language
422   \zf@v@strnum{#1}%
423   \c@zf@index\z@ \@tempswafalse
424   \loop\ifnum\c@zf@index<\@tempcntb
425     \ifnum\XeTeXOTfeaturetag\zf@basefont\c@zf@script\c@zf@language\c@zf@index=\@tempcnta
426       \@tempswatrue
427       \c@zf@index\@tempcntb
428     \else
429       \advance\c@zf@index\@ne
430     \fi
431   \repeat}

```

## 8.6 keyval definitions

This is the tedious section where we correlate all possible (eventually) font feature requests with their X<sub>Y</sub>TeX representations.

### 8.6.1 Bold/italic choosing options

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

#### Fonts

```

432 \define@key[zf]{prepare}{BoldFont}{%
433   \edef\@tempa{#1}%
434   \ifx\@tempa\@empty

```

```

435 \zf@nobftrue
436 \edef\zf@family@long{\zf@family@long nobf}%
437 \else
438 \zf@partial@fontname#1\@nil
439 \let\zf@bf\@tempa
440 \edef\zf@family@long{\zf@family@long bf:#1}%
441 \fi}
442 \define@key[zf]{preparse}{ItalicFont}{%
443 \edef\@tempa{#1}%
444 \ifx\@tempa\@empty
445 \zf@noittrue
446 \edef\zf@family@long{\zf@family@long noit}%
447 \else
448 \zf@partial@fontname#1\@nil
449 \let\zf@it\@tempa
450 \edef\zf@family@long{\zf@family@long it:#1}
451 \fi}
452 \define@key[zf]{preparse}{BoldItalicFont}{%
453 \zf@partial@fontname#1\@nil
454 \let\zf@bfit\@tempa
455 \edef\zf@family@long{\zf@family@long bfit:#1}}
456 \define@key[zf]{options}{SmallCapsFont}{%
457 \edef\@tempa{#1}%
458 \ifx\@tempa\@empty
459 \zf@nosctrue
460 \edef\zf@family@long{\zf@family@long nosc}%
461 \else
462 \zf@partial@fontname#1\@nil
463 \let\zf@sc\@tempa
464 \zf@update@family{sc:\zap@space #1 \@empty}
465 \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").

```

466 \def\zf@partial@fontname#1#2\@nil{%
467 \if#1*\relax
468 \edef\@tempa{\zf@fontname#2}%
469 \else
470 \edef\@tempa{#1#2}%
471 \fi}

```

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

```

472 \define@key[zf]{preparse}{UprightFeatures}{%
473 \def\zf@up@feat{, #1}%
474 \edef\zf@family@long{\zf@family@long rmfeat:#1}}
475 \define@key[zf]{preparse}{BoldFeatures}{%
476 \def\zf@bf@feat{, #1}%
477 \edef\zf@family@long{\zf@family@long bf@feat:#1}}

```

```

478 \define@key[zf]{preparse}{ItalicFeatures}{%
479   \def\zf@it@feat{,#1}%
480   \edef\zf@family@long{\zf@family@long itfeat:#1}}
481 \define@key[zf]{preparse}{BoldItalicFeatures}{%
482   \def\zf@bfit@feat{,#1}%
483   \edef\zf@family@long{\zf@family@long bfitfeat:#1}}
484 \define@key[zf]{options}{SmallCapsFeatures}{%
485   \unless\ifzf@firsttime\def\zf@sc@feat{,#1}\fi
486   \zf@update@family{scfeat:\zap@space #1 \@empty}}

```

### 8.6.2 The Renderer pre-parsed feature

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.

```

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

```

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

```

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

```

Exactly the same:

```

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

```

### 8.6.3 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.

```

501 \define@key[zf]{options}{Scale}{%
502   \edef\@tempa{#1}%
503   \edef\@tempb{MatchLowercase}%
504   \ifx\@tempa\@tempb
505     \zf@calc@scale{5}%
506   \else
507     \edef\@tempb{MatchUppercase}%
508     \ifx\@tempa\@tempb
509       \zf@calc@scale{8}%

```

```

510 \else
511 \edef\zf@scale{#1}%
512 \fi
513 \fi
514 \zf@update@family{+scale:\zf@scale}%
515 \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<sub>Y</sub>TeX).

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.

```

516 \newcommand\zf@calc@scale[1]{%
517 \begingroup
518 \rmfamily
519 \setlength\@tempdima{\fontdimen#1\font}%
520 \setlength\@tempdimb{\fontdimen#1\zf@basefont}%
521 \setlength\@tempdimc{1pt*\ratio{\@tempdima}{\@tempdimb}}%
522 \xdef\zf@scale{\strip@pt\@tempdimc}
523 \zf@PackageInfo{\zf@fontname\space scale = \zf@scale}%
524 \endgroup}

```

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

```

525 \define@key[zf]{options}{WordSpace}{%
526 \zf@update@family{+workspace:#1}%
527 \unless\ifzf@firsttime
528 \zf@workspace@parse#1,\zf@@ii,\zf@@iii,\zf@@
529 \fi}

```

**\zf@workspace@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}.

```

530 \def\zf@workspace@parse#1,#2,#3,#4\zf@@{%
531 \def\@tempa{#4}%
532 \ifx\@tempa\@empty
533 \setlength\@tempdima{#1\fontdimen2\zf@basefont}%
534 \@tempdimb\@tempdima
535 \@tempdimc\@tempdima
536 \else
537 \setlength\@tempdima{#1\fontdimen2\zf@basefont}%
538 \setlength\@tempdimb{#2\fontdimen3\zf@basefont}%
539 \setlength\@tempdimc{#3\fontdimen4\zf@basefont}%
540 \fi
541 \edef\zf@adjust{\zf@adjust
542 \fontdimen2\font\the\@tempdima
543 \fontdimen3\font\the\@tempdimb
544 \fontdimen4\font\the\@tempdimc}}

```

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

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

### Letterspacing

```
549 \define@key[zf]{options}{LetterSpace}{%
550   \zf@update@family{+tracking:#1}%
551   \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.

```
552 \define@key[zf]{options}{HyphenChar}{%
553   \zf@update@family{+hyphenchar:#1}%
554   \edef\@tempa{#1}%
555   \edef\@tempb{None}%
556   \ifx\@tempa\@tempb
557     \g@addto@macro\zf@adjust{\hyphenchar\font-1\relax}%
558   \else
559     \zf@check@one@char#1\zf@@
560     \ifx\@tempb\@empty
561       {\zf@basefont\expandafter\ifnum\expandafter\XeTeXcharglyph\expandafter`#1 > 0
562         \g@addto@macro\zf@adjust{%
563           {\expandafter\hyphenchar\expandafter
564             \font\expandafter`#1}}%
565       \else
566         \zf@PackageError
567           {\fontname\zf@basefont\space doesn't appear to have the glyph cor-
568            responding to #1.}
569           {You can't hyphenate with a character that's not available!}
570       \fi
571     \else
572       {\zf@basefont\ifnum\XeTeXcharglyph#1 > 0
573         \g@addto@macro\zf@adjust{\hyphenchar\font#1\relax}%
574       \else
575         \zf@PackageError
576           {\fontname\zf@basefont\space doesn't appear to have the glyph cor-
577            responding to #1.}
578           {You can't hyphenate with a character that's not available!}
579       \fi
580     \fi
581 \def\zf@check@one@char#1#2\zf@@{\def\@tempb{#2}}
```

### Colour

```
581 \define@key[zf]{options}{Colour}{%
582   \zf@update@family{+col:#1}%
```

```

583 \zf@update@ff{color=#1}}
584 \keyval@alias@key[zf]{options}{Colour}{Color}

```

## Mapping

```

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

```

### 8.6.4 Continuous font axes

```

588 \define@key[zf]{options}{Weight}{%
589 \zf@update@family{+weight:#1}%
590 \zf@update@ff{weight=#1}}
591 \define@key[zf]{options}{Width}{%
592 \zf@update@family{+width:#1}%
593 \zf@update@ff{width=#1}}
594 \define@key[zf]{options}{OpticalSize}{%
595 \ifzf@icu
596 \edef\zf@suffix{\zf@suffix/S=#1}%
597 \zf@update@family{+size:#1}
598 \fi
599 \ifzf@mm
600 \zf@update@family{+size:#1}%
601 \zf@update@ff{optical size=#1}
602 \fi
603 \ifzf@icu\else
604 \ifzf@mm\else
605 \ifzf@firsttime
606 \zf@PackageWarning
607 {\fontname\zf@basefont\space doesn't appear to have an Opti-
cal Size axis}
608 \fi
609 \fi
610 \fi}

```

### 8.6.5 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.

```

611 \zf@define@font@feature{Ligatures}
612 \zf@define@feature@option{Ligatures}{Required} {1}{0}{+rlig}
613 \zf@define@feature@option{Ligatures}{NoRequired} {1}{1}{-rlig}
614 \zf@define@feature@option{Ligatures}{Common} {1}{2}{+liga}
615 \zf@define@feature@option{Ligatures}{NoCommon} {1}{3}{-liga}
616 \zf@define@feature@option{Ligatures}{Rare} {1}{4}{+dlig}
617 \zf@define@feature@option{Ligatures}{NoRare} {1}{5}{-dlig}
618 \zf@define@feature@option{Ligatures}{Discretionary} {1}{4}{+dlig}
619 \zf@define@feature@option{Ligatures}{NoDiscretionary} {1}{5}{-dlig}
620 \zf@define@feature@option{Ligatures}{Contextual} {}{} {+clig}

```

```

621 \zf@define@feature@option{Ligatures}{NoContextual} {} {} {-clig}
622 \zf@define@feature@option{Ligatures}{Historical} {} {} {+hlig}
623 \zf@define@feature@option{Ligatures}{NoHistorical} {} {} {-hlig}
624 \zf@define@feature@option{Ligatures}{Logos} {} {} {1}{6} {}
625 \zf@define@feature@option{Ligatures}{NoLogos} {} {} {1}{7} {}
626 \zf@define@feature@option{Ligatures}{Rebus} {} {} {1}{8} {}
627 \zf@define@feature@option{Ligatures}{NoRebus} {} {} {1}{9} {}
628 \zf@define@feature@option{Ligatures}{Diphthong} {} {} {1}{10} {}
629 \zf@define@feature@option{Ligatures}{NoDiphthong} {} {} {1}{11} {}
630 \zf@define@feature@option{Ligatures}{Squared} {} {} {1}{12} {}
631 \zf@define@feature@option{Ligatures}{NoSquared} {} {} {1}{13} {}
632 \zf@define@feature@option{Ligatures}{AbbrevSquared} {} {} {1}{14} {}
633 \zf@define@feature@option{Ligatures}{NoAbbrevSquared} {} {} {1}{15} {}
634 \zf@define@feature@option{Ligatures}{Icelandic} {} {} {1}{32} {}
635 \zf@define@feature@option{Ligatures}{NoIcelandic} {} {} {1}{33} {}

```

### 8.6.6 Letters

```

636 \zf@define@font@feature{Letters}
637 \zf@define@feature@option{Letters}{Normal} {} {} {0} {}
638 \zf@define@feature@option{Letters}{Uppercase} {} {} {3}{1}{+case}
639 \zf@define@feature@option{Letters}{Lowercase} {} {} {3}{2} {}
640 \zf@define@feature@option{Letters}{SmallCaps} {} {} {3}{3}{+smcp}
641 \zf@define@feature@option{Letters}{PetiteCaps} {} {} {} {} {+pcap}
642 \zf@define@feature@option{Letters}{UppercaseSmallCaps} {} {} {} {} {+c2sc}
643 \zf@define@feature@option{Letters}{UppercasePetiteCaps} {} {} {} {} {+c2pc}
644 \zf@define@feature@option{Letters}{InitialCaps} {} {} {3}{4} {}
645 \zf@define@feature@option{Letters}{Unicase} {} {} {} {} {+unic}

```

### 8.6.7 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.

```

646 \zf@define@font@feature{Numbers}
647 \zf@define@feature@option{Numbers}{Monospaced} {} {} {6}{0}{+tnum}
648 \zf@define@feature@option{Numbers}{Proportional} {} {} {6}{1}{+pnum}
649 \zf@define@feature@option{Numbers}{Lowercase} {} {} {21}{0}{+onum}
650 \zf@define@feature@option{Numbers}{OldStyle} {} {} {21}{0}{+onum}
651 \zf@define@feature@option{Numbers}{Uppercase} {} {} {21}{1}{+lnum}
652 \zf@define@feature@option{Numbers}{Lining} {} {} {21}{1}{+lnum}
653 \zf@define@feature@option{Numbers}{SlashedZero} {} {} {14}{5}{+zero}
654 \zf@define@feature@option{Numbers}{NoSlashedZero} {} {} {14}{4}{-zero}

```

### 8.6.8 Contextuals

```

655 \zf@define@font@feature{Contextuals}
656 \zf@define@feature@option{Contextuals}{Swash} {} {} {} {} {+csw}
657 \zf@define@feature@option{Contextuals}{NoSwash} {} {} {} {} {-csw}
658 \zf@define@feature@option{Contextuals}{WordInitial} {} {} {8}{0}{+init}
659 \zf@define@feature@option{Contextuals}{NoWordInitial} {} {} {8}{1}{-init}
660 \zf@define@feature@option{Contextuals}{WordFinal} {} {} {8}{2}{+fina}

```

```

661 \zf@define@feature@option{Contextuals}{NoWordFinal}{8}{3}{-fina}
662 \zf@define@feature@option{Contextuals}{LineInitial}{8}{4}{ }
663 \zf@define@feature@option{Contextuals}{NoLineInitial}{8}{5}{ }
664 \zf@define@feature@option{Contextuals}{LineFinal}{8}{6}{+falt}
665 \zf@define@feature@option{Contextuals}{NoLineFinal}{8}{7}{-falt}
666 \zf@define@feature@option{Contextuals}{Inner}{8}{8}{+medi}
667 \zf@define@feature@option{Contextuals}{NoInner}{8}{9}{-medi}

```

### 8.6.9 Diacritics

```

668 \zf@define@font@feature{Diacritics}
669 \zf@define@feature@option{Diacritics}{Show}{9}{0}{ }
670 \zf@define@feature@option{Diacritics}{Hide}{9}{1}{ }
671 \zf@define@feature@option{Diacritics}{Decompose}{9}{2}{ }

```

### 8.6.10 Kerning

```

672 \zf@define@font@feature{Kerning}
673 \zf@define@feature@option{Kerning}{Uppercase}{ }{ }{+csp}
674 \zf@define@feature@option{Kerning}{On}{ }{ }{+kern}
675 \zf@define@feature@option{Kerning}{Off}{ }{ }{-kern}
676 %\zf@define@feature@option{Kerning}{Vertical}{ }{ }{+vkern}
677 %\zf@define@feature@option{Kerning}{VerticalAlternateProportional}{ }{ }{+vpal}
678 %\zf@define@feature@option{Kerning}{VerticalAlternateHalfWidth}{ }{ }{+vhal}

```

### 8.6.11 Vertical position

```

679 \zf@define@font@feature{VerticalPosition}
680 \zf@define@feature@option{VerticalPosition}{Normal}{10}{0}{ }
681 \zf@define@feature@option{VerticalPosition}{Superior}{10}{1}{+sup}
682 \zf@define@feature@option{VerticalPosition}{Inferior}{10}{2}{+sub}
683 \zf@define@feature@option{VerticalPosition}{ScientificInferior}{ }{ }{+sinf}
684 \zf@define@feature@option{VerticalPosition}{Ordinal}{10}{3}{+ordn}
685 \zf@define@feature@option{VerticalPosition}{Numerator}{ }{ }{+numr}
686 \zf@define@feature@option{VerticalPosition}{Denominator}{ }{ }{+dnom}

```

### 8.6.12 Fractions

```

687 \zf@define@font@feature{Fractions}
688 \zf@define@feature@option{Fractions}{On}{11}{1}{+frac}
689 \zf@define@feature@option{Fractions}{Off}{11}{0}{-frac}
690 \zf@define@feature@option{Fractions}{Diagonal}{11}{2}{ }
691 \zf@define@feature@option{Fractions}{Alternate}{ }{ }{+afrc}

```

### 8.6.13 Alternates and variants

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

```

692 \define@key[zf]{options}{Alternate}{%
693   \setkeys*[zf@feat]{Alternate}{#1}%
694   \unless\ifx\XKV@rm\@empty
695     \zf@make@feature{17}{#1}{ }%
696   \fi}

697 \define@key[zf]{options}{Variant}{%
698   \setkeys*[zf@feat]{Variant}{#1}%

```



```

699 \unless\ifx\XKV@rm\@empty
700 \edef\@tempa{\noexpand\zf@make@feature{18}{#1}{+ss@two@digits{#1}}}\@tempa
701 \fi}

```

#### 8.6.14 Style

```

702 \zf@define@font@feature{Style}
703 \zf@define@feature@option{Style}{Alternate}{+salt}
704 \zf@define@feature@option{Style}{Italic}{32}{2}{+ital}
705 \zf@define@feature@option{Style}{Ruby}{28}{2}{+ruby}
706 \zf@define@feature@option{Style}{Swash}{+swsh}
707 \zf@define@feature@option{Style}{Historic}{+hist}
708 \zf@define@feature@option{Style}{Display}{19}{1}{}
709 \zf@define@feature@option{Style}{Engraved}{19}{2}{}
710 \zf@define@feature@option{Style}{TitlingCaps}{19}{4}{+titl}
711 \zf@define@feature@option{Style}{TallCaps}{19}{5}{}
712 \zf@define@feature@option{Style}{HorizontalKana}{+hkna}
713 \zf@define@feature@option{Style}{VerticalKana}{+vkna}

```

#### 8.6.15 CJK shape

```

714 \zf@define@font@feature{CJKShape}
715 \zf@define@feature@option{CJKShape}{Traditional}{20}{0}{+trad}
716 \zf@define@feature@option{CJKShape}{Simplified}{20}{1}{+smp1}
717 \zf@define@feature@option{CJKShape}{JIS1978}{20}{2}{+jp78}
718 \zf@define@feature@option{CJKShape}{JIS1983}{20}{3}{+jp83}
719 \zf@define@feature@option{CJKShape}{JIS1990}{20}{4}{+jp90}
720 \zf@define@feature@option{CJKShape}{Expert}{20}{10}{+expt}
721 \zf@define@feature@option{CJKShape}{NLC}{20}{13}{+nlck}

```

#### 8.6.16 Character width

```

722 \zf@define@font@feature{CharacterWidth}
723 \zf@define@feature@option{CharacterWidth}{Proportional}{22}{0}{+pwid}
724 \zf@define@feature@option{CharacterWidth}{Full}{22}{1}{+fwid}
725 \zf@define@feature@option{CharacterWidth}{Half}{22}{2}{+hwid}
726 \zf@define@feature@option{CharacterWidth}{Third}{22}{3}{+twid}
727 \zf@define@feature@option{CharacterWidth}{Quarter}{22}{4}{+qwid}
728 \zf@define@feature@option{CharacterWidth}{AlternateProportional}{22}{5}{+palt}
729 \zf@define@feature@option{CharacterWidth}{AlternateHalf}{22}{6}{+halt}
730 \zf@define@feature@option{CharacterWidth}{Default}{22}{7}{}

```

#### 8.6.17 Annotation

```

731 \zf@define@font@feature{Annotation}
732 \zf@define@feature@option{Annotation}{Off}{24}{0}{-nalt}
733 \zf@define@feature@option{Annotation}{On}{+nalt}
734 \zf@define@feature@option{Annotation}{Box}{24}{1}{}
735 \zf@define@feature@option{Annotation}{RoundedBox}{24}{2}{}
736 \zf@define@feature@option{Annotation}{Circle}{24}{3}{}
737 \zf@define@feature@option{Annotation}{BlackCircle}{24}{4}{}
738 \zf@define@feature@option{Annotation}{Parenthesis}{24}{5}{}
739 \zf@define@feature@option{Annotation}{Period}{24}{6}{}
740 \zf@define@feature@option{Annotation}{RomanNumerals}{24}{7}{}

```

```

741 \zf@define@feature@option{Annotation}{Diamond}{24}{8}{}
742 \zf@define@feature@option{Annotation}{BlackSquare}{24}{9}{}
743 \zf@define@feature@option{Annotation}{BlackRoundSquare}{24}{10}{}
744 \zf@define@feature@option{Annotation}{DoubleCircle}{24}{11}{}

```

### 8.6.18 Vertical

```

745 \zf@define@font@feature{Vertical}
746 \define@key[zf@feat]{Vertical}{RotatedGlyphs}[]{}%
747 \ifzf@icu
748 \zf@make@feature{}{}{+vrt2}%
749 \else
750 \zf@update@family{+vert}%
751 \zf@update@ff{vertical}%
752 \fi}

```

### 8.6.19 Script

```

753 \newfontscript{Arabic}{arab} \newfontscript{Armenian}{armn}
754 \newfontscript{Balinese}{bali} \newfontscript{Bengali}{beng}
755 \newfontscript{Bopomofo}{bopo} \newfontscript{Braille}{brai}
756 \newfontscript{Buginese}{bugi} \newfontscript{Buhid}{buhd}
757 \newfontscript{Byzantine Music}{byzm} \newfontscript{Canadian Syllab-
ics}{cans}
758 \newfontscript{Cherokee}{cher}
759 \newfontscript{CJK Ideographic}{hani} \newfontscript{Coptic}{copt}
760 \newfontscript{Cypriot Syllabary}{cpri} \newfontscript{Cyrillic}{cyr1}
761 \newfontscript{Default}{DFLT} \newfontscript{Deseret}{dsrt}
762 \newfontscript{Devanagari}{deva} \newfontscript{Ethiopic}{ethi}
763 \newfontscript{Georgian}{geor} \newfontscript{Glagolitic}{glag}
764 \newfontscript{Gothic}{goth} \newfontscript{Greek}{grek}
765 \newfontscript{Gujarati}{gujr} \newfontscript{Gurmukhi}{guru}
766 \newfontscript{Hangul Jamo}{jamo} \newfontscript{Hangul}{hang}
767 \newfontscript{Hanunoo}{hano} \newfontscript{Hebrew}{hebr}
768 \newfontscript{Hiragana and Katakana}{kana}
769 \newfontscript{Javanese}{java} \newfontscript{Kannada}{knda}
770 \newfontscript{Kharosthi}{khar} \newfontscript{Khmer}{khmr}
771 \newfontscript{Lao}{lao} \newfontscript{Latin}{latn}
772 \newfontscript{Limbu}{limb} \newfontscript{Linear B}{linb}
773 \newfontscript{Malayalam}{mlym} \newfontscript{Math}{math}
774 \newfontscript{Mongolian}{mong}
775 \newfontscript{Musical Symbols}{musc} \newfontscript{Myanmar}{mymr}
776 \newfontscript{N'ko}{nko} \newfontscript{Ogham}{ogam}
777 \newfontscript{Old Italic}{ital} \newfontscript{Old Persian Cuneiform}{xpeo}
778 \newfontscript{Oriya}{orya} \newfontscript{Osmanya}{osma}
779 \newfontscript{Phags-pa}{phag} \newfontscript{Phoenician}{phnx}
780 \newfontscript{Runic}{runr} \newfontscript{Shavian}{shaw}
781 \newfontscript{Sinhala}{sinh} \newfontscript{Sumero-Akkadian Cuneiform}{xsux}
782 \newfontscript{Syloti Nagri}{sylo} \newfontscript{Syriac}{syr}
783 \newfontscript{Tagalog}{tglg} \newfontscript{Tagbanwa}{tagb}
784 \newfontscript{Tai Le}{tale} \newfontscript{Tai Lu}{tal}
785 \newfontscript{Tamil}{taml} \newfontscript{Telugu}{telu}
786 \newfontscript{Thaana}{thaa} \newfontscript{Thai}{thai}

```

787 \newfontscript{Tibetan}{tibt} \newfontscript{Tifinagh}{tfng}  
 788 \newfontscript{Ugaritic Cuneiform}{ugar}\newfontscript{Yi}{yi }

## 8.6.20 Language

789 \newfontlanguage{Abaza}{ABA}\newfontlanguage{Abkhazian}{ABK}\newfontlanguage{Adyghe}{ADY}  
 790 \newfontlanguage{Afrikaans}{AFK}\newfontlanguage{Afar}{AFR}\newfontlanguage{Agaw}{AGW}  
 791 \newfontlanguage{Altai}{ALT}\newfontlanguage{Amharic}{AMH}\newfontlanguage{Arabic}{ARA}  
 792 \newfontlanguage{Aari}{ARI}\newfontlanguage{Arakanese}{ARK}\newfontlanguage{Assamese}{ASM}  
 793 \newfontlanguage{Athapaskan}{ATH}\newfontlanguage{Avar}{AVR}\newfontlanguage{Awadhi}{AWA}  
 794 \newfontlanguage{Aymara}{AYM}\newfontlanguage{Azeri}{AZE}\newfontlanguage{Badaga}{BAD}  
 795 \newfontlanguage{Baghelkhandi}{BAG}\newfontlanguage{Balkar}{BAL}\newfontlanguage{Bauile}{BAU}  
 796 \newfontlanguage{Berber}{BBR}\newfontlanguage{Bench}{BCH}\newfontlanguage{Bible Cree}{BCR}  
 797 \newfontlanguage{Belarussian}{BEL}\newfontlanguage{Bemba}{BEM}\newfontlanguage{Bengali}{BEN}  
 798 \newfontlanguage{Bulgarian}{BGR}\newfontlanguage{Bhili}{BHI}\newfontlanguage{Bhojpurī}{BHO}  
 799 \newfontlanguage{Bikol}{BIK}\newfontlanguage{Bilen}{BIL}\newfontlanguage{Blackfoot}{BKF}  
 800 \newfontlanguage{Balochi}{BLI}\newfontlanguage{Balante}{BLN}\newfontlanguage{Balti}{BLT}  
 801 \newfontlanguage{Bambara}{BMB}\newfontlanguage{Bamileke}{BML}\newfontlanguage{Breton}{BRE}  
 802 \newfontlanguage{Brahui}{BRH}\newfontlanguage{Braj Bhasha}{BRI}\newfontlanguage{Burmese}{BRM}  
 803 \newfontlanguage{Bashkir}{BSH}\newfontlanguage{Beti}{BTI}\newfontlanguage{Catalan}{CAT}  
 804 \newfontlanguage{Cebuano}{CEB}\newfontlanguage{Chechen}{CHE}\newfontlanguage{Chaha Gurage}{CHG}  
 805 \newfontlanguage{Chattisgarhi}{CHH}\newfontlanguage{Chichewa}{CHI}\newfontlanguage{Chukchi}{CHK}  
 806 \newfontlanguage{Chipewyan}{CHP}\newfontlanguage{Cherokee}{CHR}\newfontlanguage{Chuvash}{CHU}  
 807 \newfontlanguage{Comorian}{CMR}\newfontlanguage{Coptic}{COP}\newfontlanguage{Cree}{CRE}  
 808 \newfontlanguage{Carrier}{CRR}\newfontlanguage{Crimean Tatar}{CRT}\newfontlanguage{Church Slavonic}{CHS}  
 809 \newfontlanguage{Czech}{CSY}\newfontlanguage{Danish}{DAN}\newfontlanguage{Dargwa}{DAR}  
 810 \newfontlanguage{Woods Cree}{DCR}\newfontlanguage{German}{DEU}\newfontlanguage{Default}{DFLT}  
 811 \newfontlanguage{Dogri}{DGR}\newfontlanguage{Divehi}{DIV}\newfontlanguage{Djerma}{DJR}  
 812 \newfontlanguage{Dangme}{DNG}\newfontlanguage{Dinka}{DNK}\newfontlanguage{Dungan}{DUN}  
 813 \newfontlanguage{Dzongkha}{DZN}\newfontlanguage{Ebirā}{EBI}\newfontlanguage{Eastern Cree}{ECR}  
 814 \newfontlanguage{Edo}{EDO}\newfontlanguage{Efik}{EFI}\newfontlanguage{Greek}{ELL}  
 815 \newfontlanguage{English}{ENG}\newfontlanguage{Erzya}{ERZ}\newfontlanguage{Spanish}{ESP}  
 816 \newfontlanguage{Estonian}{ETI}\newfontlanguage{Basque}{EUQ}\newfontlanguage{Evenki}{EVK}  
 817 \newfontlanguage{Even}{EVN}\newfontlanguage{Ewe}{EWE}\newfontlanguage{French Antillean}{FAN}  
 818 \newfontlanguage{Farsi}{FAR}\newfontlanguage{Finnish}{FIN}\newfontlanguage{Fijian}{FJI}  
 819 \newfontlanguage{Flemish}{FLE}\newfontlanguage{Forest Nenets}{FNE}\newfontlanguage{Fon}{FON}  
 820 \newfontlanguage{Faroese}{FOS}\newfontlanguage{French}{FRA}\newfontlanguage{Frisian}{FRI}  
 821 \newfontlanguage{Friulian}{FRL}\newfontlanguage{Futa}{FTA}\newfontlanguage{Fulani}{FUL}  
 822 \newfontlanguage{Ga}{GAD}\newfontlanguage{Gaelic}{GAE}\newfontlanguage{Gagauz}{GAG}  
 823 \newfontlanguage{Galician}{GAL}\newfontlanguage{Garshuni}{GAR}\newfontlanguage{Garhwali}{GAW}  
 824 \newfontlanguage{Ge'ez}{GEZ}\newfontlanguage{Gilyak}{GIL}\newfontlanguage{Gumuz}{GMZ}  
 825 \newfontlanguage{Gondi}{GON}\newfontlanguage{Greenlandic}{GRN}\newfontlanguage{Garō}{GRO}  
 826 \newfontlanguage{Guarani}{GUA}\newfontlanguage{Gujarati}{GUJ}\newfontlanguage{Haitian}{HAI}  
 827 \newfontlanguage{Halam}{HAL}\newfontlanguage{Harauti}{HAR}\newfontlanguage{Hausa}{HAU}  
 828 \newfontlanguage{Hawaiian}{HAW}\newfontlanguage{Hammer-Banna}{HBN}\newfontlanguage{Hiligaynon}{HIL}  
 829 \newfontlanguage{Hindi}{HIN}\newfontlanguage{High Mari}{HMA}\newfontlanguage{Hindko}{HND}  
 830 \newfontlanguage{Ho}{HO}\newfontlanguage{Harari}{HRI}\newfontlanguage{Croatian}{HRV}  
 831 \newfontlanguage{Hungarian}{HUN}\newfontlanguage{Armenian}{HYE}\newfontlanguage{Igbo}{IBO}  
 832 \newfontlanguage{Ijo}{IJO}\newfontlanguage{Ilokano}{ILO}\newfontlanguage{Indonesian}{IND}  
 833 \newfontlanguage{Ingush}{ING}\newfontlanguage{Inuktitut}{INU}\newfontlanguage{Irish}{IRI}  
 834 \newfontlanguage{Irish Traditional}{IRT}\newfontlanguage{Icelandic}{ISL}\newfontlanguage{Inari Sami}{INR}

835 \newfontlanguage{Italian}{ITA}\newfontlanguage{Hebrew}{IWR}\newfontlanguage{Javanese}{JAV}  
836 \newfontlanguage{Yiddish}{JII}\newfontlanguage{Japanese}{JAN}\newfontlanguage{Judezmo}{JUD}  
837 \newfontlanguage{Jula}{JUL}\newfontlanguage{Kabardian}{KAB}\newfontlanguage{Kachchi}{KAC}  
838 \newfontlanguage{Kalenjin}{KAL}\newfontlanguage{Kannada}{KAN}\newfontlanguage{Karachay}{KAR}  
839 \newfontlanguage{Georgian}{KAT}\newfontlanguage{Kazakh}{KAZ}\newfontlanguage{Kebena}{KEB}  
840 \newfontlanguage{Khutsuri Georgian}{KGE}\newfontlanguage{Khakass}{KHA}\newfontlanguage{Khanty-  
Kazim}{KHK}  
841 \newfontlanguage{Khmer}{KHM}\newfontlanguage{Khanty-Shurishkar}{KHS}\newfontlanguage{Khanty-  
Vakhi}{KHV}  
842 \newfontlanguage{Khowar}{KHW}\newfontlanguage{Kikuyu}{KIK}\newfontlanguage{Kirghiz}{KIR}  
843 \newfontlanguage{Kisii}{KIS}\newfontlanguage{Kokni}{KKN}\newfontlanguage{Kalmyk}{KLM}  
844 \newfontlanguage{Kamba}{KMB}\newfontlanguage{Kumaoni}{KMN}\newfontlanguage{Komo}{KMO}  
845 \newfontlanguage{Komso}{KMS}\newfontlanguage{Kanuri}{KNR}\newfontlanguage{Kodagu}{KOD}  
846 \newfontlanguage{Korean Old Hangul}{KOH}\newfontlanguage{Konkani}{KOK}\newfontlanguage{Kikongo}  
847 \newfontlanguage{Komi-Permyak}{KOP}\newfontlanguage{Korean}{KOR}\newfontlanguage{Komi-  
Zyrian}{KOZ}  
848 \newfontlanguage{Kpelle}{KPL}\newfontlanguage{Krio}{KRI}\newfontlanguage{Karakalpak}{KRK}  
849 \newfontlanguage{Karelian}{KRL}\newfontlanguage{Karaim}{KRM}\newfontlanguage{Karen}{KRN}  
850 \newfontlanguage{Koorote}{KRT}\newfontlanguage{Kashmiri}{KSH}\newfontlanguage{Khasi}{KSI}  
851 \newfontlanguage{Kildin Sami}{KSM}\newfontlanguage{Kui}{KUI}\newfontlanguage{Kulvi}{KUL}  
852 \newfontlanguage{Kumyk}{KUM}\newfontlanguage{Kurdish}{KUR}\newfontlanguage{Kurukh}{KUU}  
853 \newfontlanguage{Kuy}{KUY}\newfontlanguage{Koryak}{KYK}\newfontlanguage{Ladin}{LAD}  
854 \newfontlanguage{Lahuli}{LAH}\newfontlanguage{Lak}{LAK}\newfontlanguage{Lambani}{LAM}  
855 \newfontlanguage{Lao}{LAO}\newfontlanguage{Latin}{LAT}\newfontlanguage{Laz}{LAZ}  
856 \newfontlanguage{L-Cree}{LCR}\newfontlanguage{Ladakhi}{LDK}\newfontlanguage{Lezgi}{LEZ}  
857 \newfontlanguage{Lingala}{LIN}\newfontlanguage{Low Mari}{LMA}\newfontlanguage{Limbu}{LMB}  
858 \newfontlanguage{Lomwe}{LMW}\newfontlanguage{Lower Sorbian}{LSB}\newfontlanguage{Lule Sami}{LSM}  
859 \newfontlanguage{Lithuanian}{LTH}\newfontlanguage{Luba}{LUB}\newfontlanguage{Luganda}{LUG}  
860 \newfontlanguage{Luhya}{LUH}\newfontlanguage{Luo}{LUO}\newfontlanguage{Latvian}{LVI}  
861 \newfontlanguage{Majang}{MAJ}\newfontlanguage{Makua}{MAK}\newfontlanguage{Malayalam Tra-  
ditional}{MAL}  
862 \newfontlanguage{Mansi}{MAN}\newfontlanguage{Marathi}{MAR}\newfontlanguage{Marwari}{MAW}  
863 \newfontlanguage{Mbundu}{MBN}\newfontlanguage{Manchu}{MCH}\newfontlanguage{Moose Cree}{MCR}  
864 \newfontlanguage{Mende}{MDE}\newfontlanguage{Me'en}{MEN}\newfontlanguage{Mizo}{MIZ}  
865 \newfontlanguage{Macedonian}{MKD}\newfontlanguage{Male}{MLE}\newfontlanguage{Malagasy}{MLG}  
866 \newfontlanguage{Malinke}{MLN}\newfontlanguage{Malayalam Reformed}{MLR}\newfontlanguage{Malay}  
867 \newfontlanguage{Mandinka}{MND}\newfontlanguage{Mongolian}{MNG}\newfontlanguage{Manipuri}{MNI}  
868 \newfontlanguage{Maninka}{MNK}\newfontlanguage{Manx Gaelic}{MNX}\newfontlanguage{Moksha}{MOK}  
869 \newfontlanguage{Moldavian}{MOL}\newfontlanguage{Mon}{MON}\newfontlanguage{Moroccan}{MOR}  
870 \newfontlanguage{Maori}{MRI}\newfontlanguage{Maithili}{MTH}\newfontlanguage{Maltese}{MTS}  
871 \newfontlanguage{Mundari}{MUN}\newfontlanguage{Naga-Assamese}{NAG}\new-  
fontlanguage{Nanai}{NAN}  
872 \newfontlanguage{Naskapi}{NAS}\newfontlanguage{N-Cree}{NCR}\newfontlanguage{Ndebele}{NDB}  
873 \newfontlanguage{Ndonga}{NDG}\newfontlanguage{Nepali}{NEP}\newfontlanguage{Newari}{NEW}  
874 \newfontlanguage{Nagari}{NGR}\newfontlanguage{Norway House Cree}{NHC}\new-  
fontlanguage{Nisi}{NIS}  
875 \newfontlanguage{Niuean}{NIU}\newfontlanguage{Nkole}{NKL}\newfontlanguage{N'ko}{NKO}  
876 \newfontlanguage{Dutch}{NLD}\newfontlanguage{Nogai}{NOG}\newfontlanguage{Norwegian}{NOR}  
877 \newfontlanguage{Northern Sami}{NSM}\newfontlanguage{Northern Tai}{NTA}\new-  
fontlanguage{Esperanto}{NTO}

878 \newfontlanguage{Nynorsk}{NYN} \newfontlanguage{Oji-Cree}{OCR} \newfont-  
 language{Ojibway}{OJB}  
 879 \newfontlanguage{Oriya}{ORI} \newfontlanguage{Oromo}{ORO} \newfontlanguage{Ossetian}{OSS}  
 880 \newfontlanguage{Palestinian Aramaic}{PAA} \newfontlanguage{Pali}{PAL} \new-  
 fontlanguage{Punjabi}{PAN}  
 881 \newfontlanguage{Palpa}{PAP} \newfontlanguage{Pashto}{PAS} \newfontlanguage{Polytonic Greek}{PGR}  
 882 \newfontlanguage{Pilipino}{PIL} \newfontlanguage{Palaung}{PLG} \newfont-  
 language{Polish}{PLK}  
 883 \newfontlanguage{Provencal}{PRO} \newfontlanguage{Portuguese}{PTG} \new-  
 fontlanguage{Chin}{QIN}  
 884 \newfontlanguage{Rajasthani}{RAJ} \newfontlanguage{R-Cree}{RCR} \newfont-  
 language{Russian Buriat}{RBU}  
 885 \newfontlanguage{Riang}{RIA} \newfontlanguage{Rhaeto-Romanic}{RMS} \new-  
 fontlanguage{Romanian}{ROM}  
 886 \newfontlanguage{Romany}{ROY} \newfontlanguage{Rusyn}{RSY} \newfontlanguage{Ruanda}{RUA}  
 887 \newfontlanguage{Russian}{RUS} \newfontlanguage{Sadri}{SAD} \newfontlanguage{Sanskrit}{SAN}  
 888 \newfontlanguage{Santali}{SAT} \newfontlanguage{Sayisi}{SAY} \newfontlanguage{Sekota}{SEK}  
 889 \newfontlanguage{Selkup}{SEL} \newfontlanguage{Sango}{SGO} \newfontlanguage{Shan}{SHN}  
 890 \newfontlanguage{Sibe}{SIB} \newfontlanguage{Sidamo}{SID} \newfontlanguage{Silte Gurage}{SIG}  
 891 \newfontlanguage{Skolt Sami}{SKS} \newfontlanguage{Slovak}{SKY} \newfont-  
 language{Slavey}{SLA}  
 892 \newfontlanguage{Slovenian}{SLV} \newfontlanguage{Somali}{SML} \newfont-  
 language{Samoan}{SMO}  
 893 \newfontlanguage{Sena}{SNA} \newfontlanguage{Sindhi}{SND} \newfontlanguage{Sinhalese}{SNH}  
 894 \newfontlanguage{Soninke}{SNK} \newfontlanguage{Sodo Gurage}{SOG} \new-  
 fontlanguage{Sotho}{SOT}  
 895 \newfontlanguage{Albanian}{SQI} \newfontlanguage{Serbian}{SRB} \newfont-  
 language{Saraiki}{SRK}  
 896 \newfontlanguage{Serer}{SRR} \newfontlanguage{South Slavey}{SSL} \newfont-  
 language{Southern Sami}{SSM}  
 897 \newfontlanguage{Suri}{SUR} \newfontlanguage{Svan}{SVA} \newfontlanguage{Swedish}{SVE}  
 898 \newfontlanguage{Swadaya Aramaic}{SWA} \newfontlanguage{Swahili}{SWK} \new-  
 fontlanguage{Swazi}{SWZ}  
 899 \newfontlanguage{Sutu}{SXT} \newfontlanguage{Syriac}{SYR} \newfontlanguage{Tabasaran}{TAB}  
 900 \newfontlanguage{Tajiki}{TAJ} \newfontlanguage{Tamil}{TAM} \newfontlanguage{Tatar}{TAT}  
 901 \newfontlanguage{TH-Cree}{TCR} \newfontlanguage{Telugu}{TEL} \newfontlanguage{Tongan}{TGN}  
 902 \newfontlanguage{Tigre}{TGR} \newfontlanguage{Tigrinya}{TGY} \newfontlanguage{Thai}{THA}  
 903 \newfontlanguage{Tahitian}{THT} \newfontlanguage{Tibetan}{TIB} \newfont-  
 language{Turkmen}{TKM}  
 904 \newfontlanguage{Temne}{TMN} \newfontlanguage{Tswana}{TNA} \newfontlanguage{Tundra Nenets}{TNE}  
 905 \newfontlanguage{Tonga}{TNG} \newfontlanguage{Todo}{TOD}  
 906 \newfontlanguage{Tsonga}{TSG} \newfontlanguage{Turoyo Aramaic}{TUA} \new-  
 fontlanguage{Tulu}{TUL}  
 907 \newfontlanguage{Tuvini}{TUV} \newfontlanguage{Twili}{TWI} \newfontlanguage{Udmurt}{UDM}  
 908 \newfontlanguage{Ukrainian}{UKR} \newfontlanguage{Urdu}{URD} \newfontlanguage{Upper Sor-  
 bian}{USB}  
 909 \newfontlanguage{Uyghur}{UYG} \newfontlanguage{Uzbek}{UZB} \newfontlanguage{Venda}{VEN}  
 910 \newfontlanguage{Vietnamese}{VIT} \newfontlanguage{Wa}{WA} \newfontlanguage{Wagdi}{WAG}  
 911 \newfontlanguage{West-Cree}{WCR} \newfontlanguage{Welsh}{WEL} \newfontlanguage{Wolof}{WLF}  
 912 \newfontlanguage{Tai Lue}{XBD} \newfontlanguage{Xhosa}{XHS} \newfontlanguage{Yakut}{YAK}

```

913 \newfontlanguage{Yoruba}{YBA} \newfontlanguage{Y-Cree}{YCR} \newfontlanguage{Yi Clas-
    sic}{YIC}
914 \newfontlanguage{Yi Modern}{YIM} \newfontlanguage{Chinese Hong Kong}{ZHH}
915 \newfontlanguage{Chinese Phonetic}{ZHP} \newfontlanguage{Chinese Simpli-
    fied}{ZHS}
916 \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:

```

917 \define@key[zf@feat]{Lang}{Turkish}[]{}%
918   \zf@check@ot@lang{TRK}%
919   \if@tempswa
920     \c@zf@language\@tempcnta\relax
921     \xdef\zf@language@name{Turkish}%
922     \xdef\zf@family@long{\zf@family@long+lang=Turkish}%
923     \xdef\zf@pre@ff{\zf@pre@ff language=TRK,}%
924   \else
925     \zf@check@ot@lang{TUR}%
926     \if@tempswa
927       \c@zf@language\@tempcnta\relax
928       \xdef\zf@language@name{Turkish}%
929       \xdef\zf@family@long{\zf@family@long+lang=Turkish}%
930       \xdef\zf@pre@ff{\zf@pre@ff language=TUR,}%
931     \else
932       \zf@PackageWarning{Font \fontname\zf@basefont does not contain
933         language '#1' for script '\zf@script@name'}%
934     \fi
935   \fi}

```

## 8.7 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).

```

\sisshape First, the commands for actually selecting italic small caps are defined. I use si
\textsi 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.
936 \providecommand*\sidefault{si}
937 \DeclareRobustCommand\sisshape{}%
938   \not@math@alphabet\sisshape\relax
939   \fontshape\sidefault\selectfont}
940 \DeclareTextFontCommand\textsi{\sisshape}

\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.
941 \newcommand*\zf@merge@shape[3]{}%
942   \edef\@tempa{#1}%

```

```

943 \edef\@tempb{#2}%
944 \ifx\f@shape\@tempb
945   \ifcsname\f@encoding/\f@family/\f@series/#3\endcsname
946     \edef\@tempa{#3}%
947   \fi
948 \fi
949 \fontshape{\@tempa}\selectfont}

```

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

```

\upshape 950 \DeclareRobustCommand{\itshape}{%
951   \not@math@alphabet\itshape\mathit
952   \zf@merge@shape\itdefault\scdefault\sidefault}
953 \DeclareRobustCommand{\slshape}{%
954   \not@math@alphabet\slshape\relax
955   \zf@merge@shape\sldefault\scdefault\sidefault}
956 \DeclareRobustCommand{\scshape}{%
957   \not@math@alphabet\scshape\relax
958   \zf@merge@shape\scdefault\itdefault\sidefault}
959 \DeclareRobustCommand{\upshape}{%
960   \not@math@alphabet\upshape\relax
961   \zf@merge@shape\updefault\sidefault\scdefault}

```

\em Redefinitions moved to the xltextra package.  
\emph

## 8.8 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<sub>Y</sub>TeX 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.

```

962 \@ifpackageloaded{euler}{\zf@euler@package@loadedtrue}
963                          {\zf@euler@package@loadedfalse}
964 \AtBeginDocument{%
965   \let\zf@font@warning\@font@warning
966   \let\@font@warning\@font@info
967   \@ifpackageloaded{euler}{%
968     \ifzf@euler@package@loaded
969       \zf@math@eulertrue
970     \else
971       \zf@PackageError{The euler package must be loaded BEFORE fontspec}

```

```

972      {fontspec only overwrites euler's attempt to \MessageBreak
973      define the maths text fonts if fontspec is \MessageBreak
974      loaded after euler. Type <return> to proceed\MessageBreak
975      with incorrect \protect\mathit, \protect\mathbf, etc}
976      \fi}{}
977 \ifpackageloaded{lucbmath}{\zf@math@lucidatrue}{}
978 \ifpackageloaded{lucidabr}{\zf@math@lucidatrue}{}
979 \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<sup>A</sup>T<sub>E</sub>X'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.)

```

980 \DeclareSymbolFont{legacymaths}{OT1}{cmr}{m}{n}
981 \SetSymbolFont{legacymaths}{bold}{OT1}{cmr}{bx}{n}
982 \DeclareMathAccent{\acute}{\mathalpha}{legacymaths}{19}
983 \DeclareMathAccent{\grave}{\mathalpha}{legacymaths}{18}
984 \DeclareMathAccent{\ddot}{\mathalpha}{legacymaths}{127}
985 \DeclareMathAccent{\tilde}{\mathalpha}{legacymaths}{126}
986 \DeclareMathAccent{\bar}{\mathalpha}{legacymaths}{22}
987 \DeclareMathAccent{\breve}{\mathalpha}{legacymaths}{21}
988 \DeclareMathAccent{\check}{\mathalpha}{legacymaths}{20}
989 \DeclareMathAccent{\hat}{\mathalpha}{legacymaths}{94} % too bad, euler
990 \DeclareMathAccent{\dot}{\mathalpha}{legacymaths}{95}
991 \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-\thinspace\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}
```

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

```
992 \begingroup
993   \mathchardef\@tempa="603A %
994   \let\next\egroup
995   \ifx\colon\@tempa
996     \DeclareMathSymbol{\colon}{\mathpunct}{legacymaths}{58}
997   \fi
998 \endgroup
```

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

```
999 \ifzf@math@euler\else
1000 \DeclareMathSymbol{!}{\mathclose}{legacymaths}{33}
1001 \DeclareMathSymbol{:}{\mathrel}{legacymaths}{58}
1002 \DeclareMathSymbol{;}{\mathpunct}{legacymaths}{59}
1003 \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.

```
1004 \ifzf@math@lucida\else
1005   \DeclareMathSymbol{0}{\mathalpha}{legacymaths}{`0}
1006   \DeclareMathSymbol{1}{\mathalpha}{legacymaths}{`1}
1007   \DeclareMathSymbol{2}{\mathalpha}{legacymaths}{`2}
1008   \DeclareMathSymbol{3}{\mathalpha}{legacymaths}{`3}
1009   \DeclareMathSymbol{4}{\mathalpha}{legacymaths}{`4}
1010   \DeclareMathSymbol{5}{\mathalpha}{legacymaths}{`5}
1011   \DeclareMathSymbol{6}{\mathalpha}{legacymaths}{`6}
1012   \DeclareMathSymbol{7}{\mathalpha}{legacymaths}{`7}
1013   \DeclareMathSymbol{8}{\mathalpha}{legacymaths}{`8}
1014   \DeclareMathSymbol{9}{\mathalpha}{legacymaths}{`9}
1015   \DeclareMathSymbol{\Gamma}{\mathalpha}{legacymaths}{0}
1016   \DeclareMathSymbol{\Delta}{\mathalpha}{legacymaths}{1}
1017   \DeclareMathSymbol{\Theta}{\mathalpha}{legacymaths}{2}
1018   \DeclareMathSymbol{\Lambda}{\mathalpha}{legacymaths}{3}
1019   \DeclareMathSymbol{\Xi}{\mathalpha}{legacymaths}{4}
1020   \DeclareMathSymbol{\Pi}{\mathalpha}{legacymaths}{5}
1021   \DeclareMathSymbol{\Sigma}{\mathalpha}{legacymaths}{6}
1022   \DeclareMathSymbol{\Upsilon}{\mathalpha}{legacymaths}{7}
1023   \DeclareMathSymbol{\Phi}{\mathalpha}{legacymaths}{8}
1024   \DeclareMathSymbol{\Psi}{\mathalpha}{legacymaths}{9}
1025   \DeclareMathSymbol{\Omega}{\mathalpha}{legacymaths}{10}
1026   \DeclareMathSymbol{+}{\mathbin}{legacymaths}{43}
1027   \DeclareMathSymbol{=}{\mathrel}{legacymaths}{61}
1028   \DeclareMathDelimiter{\}{\mathopen}{legacymaths}{40}{largesymbols}{0}
1029   \DeclareMathDelimiter{)}{\mathclose}{legacymaths}{41}{largesymbols}{1}
1030   \DeclareMathDelimiter{[}{\mathopen}{legacymaths}{91}{largesymbols}{2}
1031   \DeclareMathDelimiter{]}{\mathclose}{legacymaths}{93}{largesymbols}{3}
```

```

1032 \DeclareMathDelimiter{/}{\mathord}{\legacymaths}{47}{\largesymbols}{14}
1033 \DeclareMathSymbol{\mathdollar}{\mathord}{\legacymaths}{36}
1034 \fi
1035 \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<sup>A</sup>T<sub>E</sub>X 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`.

```

1036 \DeclareSymbolFont{operators}\zf@enc\zf@rmmaths\mddefault\updefault
1037 \SetSymbolFont{operators}{normal}\zf@enc\zf@rmmaths\mddefault\updefault
1038 \SetMathAlphabet\mathrm{normal}\zf@enc\zf@rmmaths\mddefault\updefault
1039 \SetMathAlphabet\mathit{normal}\zf@enc\zf@rmmaths\mddefault\itdefault
1040 \SetMathAlphabet\mathbf{normal}\zf@enc\zf@rmmaths\bfdefault\updefault
1041 \SetMathAlphabet\mathsf{normal}\zf@enc\zf@sffmaths\mddefault\updefault
1042 \SetMathAlphabet\mathtt{normal}\zf@enc\zf@ttmaths\mddefault\updefault
1043 \SetSymbolFont{operators}{bold}\zf@enc\zf@rmmaths\bfdefault\updefault
1044 \ifdefined\zf@rmboldmaths
1045 \SetMathAlphabet\mathrm{bold}\zf@enc\zf@rmboldmaths\mddefault\updefault
1046 \SetMathAlphabet\mathbf{bold}\zf@enc\zf@rmboldmaths\bfdefault\updefault
1047 \SetMathAlphabet\mathit{bold}\zf@enc\zf@rmboldmaths\mddefault\itdefault
1048 \else
1049 \SetMathAlphabet\mathrm{bold}\zf@enc\zf@rmmaths\bfdefault\updefault
1050 \SetMathAlphabet\mathit{bold}\zf@enc\zf@rmmaths\bfdefault\itdefault
1051 \fi
1052 \SetMathAlphabet\mathsf{bold}\zf@enc\zf@sffmaths\bfdefault\updefault
1053 \SetMathAlphabet\mathtt{bold}\zf@enc\zf@ttmaths\bfdefault\updefault
1054 \let\font@warning\zf@font@warning}

```

## 8.9 Option processing

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

```

1055 \DeclareOption{config}{%
1056 \InputIfFileExists{fontspec.cfg}
1057 {\typeout{fontspec.cfg loaded.}}
1058 {\typeout{fontspec.cfg would be loaded now if it existed.}}}
1059 \DeclareOption{noconfig}{}
1060 \DeclareOption{quiet}{\let\zf@PackageWarning\zf@PackageInfo}
1061 \ExecuteOptions{config}
1062 \ProcessOptions

```

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.tex

```
1 %!TEX TS-program =xelatex
2 \documentclass[12pt]{article}
3
4 \usepackage{euler,fontspec,graphicx}
5
6 \defaultfontfeatures{Scale=MatchLowercase,Mapping=tex-text}
7 \setromanfont{Hoefler Text}
8 \setsansfont{Gill Sans}
9 \setmonofont{Lucida Sans Typewriter}
10
11 %% Define the \XeTeX logo:
12 \DeclareRobustCommand\XeTeX{%
```

```

13 \mbox{\smash{%
14   X\lower.5ex\hbox{\kern-.12em\reflectbox{E}}\kern-.1667em
15   T\kern-.1667em\lower.5ex\hbox{E}}\kern-.12em X}}\@}
16%% The logo should be defined on a per-document basis
17%% so that its parameters may be fine tuned for the fonts used.
18
19\begin{document}
20\pagestyle{empty}
21
22\section{The basics of the \textsf{fontspec} package}
23
24The \textsf{fontspec} package enables automatic font selection for \La-
    TeX{} documents typeset with \XeTeX{}. The basic command is\
25\indent \verb|\fontspec[font features]{Mac OS X font display name}|.\
26As an example:
27
28\begin{center}
29  \Large
30  \fontspec[
31    Colour          = 0000CC,
32    Numbers          = OldStyle,
33    VerticalPosition = Ordinal,
34    Variant          = 2
35    ]{Apple Chancery}
36  My 1st example of Apple Chancery
37\end{center}
38
39The default roman, sans serif, and typewriter fonts may be set with the \verb|\setromanfont|, \verb|
    mands, respectively, as shown in the preamble. They take the same syn-
    tax as the \verb|\fontspec| package. All expected font shapes are available:
40
41\begin{center}
42  {\scshape Small caps and \itshape small caps italic\dots}\
43  {\sffamily\bfseries Bold sans serif and \itshape bold italic sans serif\dots}
44\end{center}
45
46With 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---note that the \textsf{eulervm} package will not work in \Xe-
    TeX{} because it uses virtual fonts), since the default Computer Mod-
    ern maths font is rather light.
47\[
48  \mathcal{F}(s) = \int\limits_0^\infty f(t) e^{-st}\mathrm{d}t
49\]
50
51You'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.

```

```
52
53 Please see the documentation for font feature explanation and further pack-
    age niceties.
54
55 \end{document}
```

## Change History

v1.0	
General: Initial version.	26
v1.1	
General: Name change to fontspec.	26
\setromanfont: Implemented (with friends).	28
v1.10	
General: Color brought back into the .sty	44
New feature LetterSpace.	44
Some babel encoding problems resolved.	27
\addfontfeatures: Saved family information macro changes.	29
\AtBeginDocument: Added lucimatx checking. (Not really tested, though.)	54
Fixed Lucida bug (missing \else)	54
\zf@fontspec: Saved family info split into two (now three) macros.	34
Space zapped from L <sup>A</sup> T <sub>E</sub> X family name due to various problems.	34
\zf@make@feature: Removed embarrassing space after warnings.	37
v1.11	
General: HyphenChar checks its input now.	44
Added better support for Turkish language selection.	53
Ensure bold/italic fonts are loaded with the same renderer as the regular	
font even if unspecified.	34
OpenType Variant fixed.	47
\emph: Redefinitions moved to xltextra.	53
\newfontface: Name change from \newfontfamily.	29
\newfontlanguage: Fixed \c@zf@language setting not being global.	31
\newfontscript: Fixed \c@zf@script setting not being global.	31
\zf@wordspace@parse: Improved saving \fontdimen stuff to \zd@adjust(also	
see PunctuationSpace).	43
v1.2	
General: Initial OpenType support.	26
Support for Scale.	42
v1.3	
General: More OpenType support.	26
Support for Mapping and Colour.	44
\defaultfontfeatures: Implemented.	29
\newAATfeature: Implemented.	30
\newfontfeature: Implemented.	30
v1.3a	
General: Bug fix for OpenType small caps.	46
v1.4	
General: Support for Weight and Width AAT features.	45
\AtBeginDocument: Selects the default \mathXX fonts.	54
\defaultfontfeatures: Name changed from \setdefaultoptions.	29
v1.5	
General: New options for arbitrary bold/italic shapes.	40
\addfontfeatures: Implemented.	29
\zf@fontspec: Added code for choosing arbitrary bold/italic fonts.	34
Checks if the font family has already been defined.	34
NFSS specifiers now take the default values.	34
\zf@make@font@shapes: Absorbed font-checking from \zf@fontspec.	35

v1.5a	
\AtBeginDocument: Added fix for Computer Modern maths.	54
v1.6	
General: Bold option aliased to BoldFont.	40
LetterCase is now Letters and options changed appropriately.	46
Scale feature now updates family name.	42
All AAT Fractions features offered.	47
New OpenType feature: Language	49
New OpenType feature: Script	49
OpenType letters features: PetiteCaps and PETITECAPS.	46
OpenType ligature features: Contextual and Historical.	45
OpenType stylistic sets supports under the Variant option.	47
\addfontfeatures: Removed \relaxing of temporary macros.	29
\AtBeginDocument: Removed mathtime support since XeTeX doesn't handle virtual fonts. Why did I put it in in the first place?	54
\fontspec: Removed \zf@currfont (unnecessary)	27
\newfontface: Implemented.	29
\newfontfeature: newff counter now uses LaTeX methods rather than primitive TeX. I don't know if there is any advantage to this.	30
\setromanfont: Changed \rmdefault, etc., assigning to use \let directly.	28
\zf@fontspec: Added code for choosing arbitrary bold/italic font features.	34
Writes some info to the .log file	34
\zf@get@feature@requests: Removed the space between the comma and \zf@options when it's concatenated with the defaults.	36
v1.7	
General: Style feature renamed from StyleOptions.	47
AAT Numbers:SlashedZero.	46
New feature: Annotation	48
New feature: CharacterShape	48
New feature: CharacterWidth	48
New feature: OpticalSize; works with both OpenType and MM fonts.	45
OpenType Alternate Fractions feature.	47
OpenType Alternate now only AAT.	47
Removed AAT check for weight/width axes (could also be Multiple Master)	45
\zf@define@feature@option: Implemented for the bulk of the feature processing code.	37
\zf@fontspec: Optional argument now mandatory.	34
\zf@make@aat@feature@string: Changed some \edefs to \let	38
Removed third argument; always saves the feature string in \zf@thisfontfeature	38
\zf@make@feature: Accommodation of the \zf@thisfontfeature change.	37
\zf@make@font@shapes: Changed some \edefs to \let.	35
Support for the OpticalSize feature.	35
\zf@make@smallcaps: Accommodation of the \zf@thisfontfeature change.	37
\zf@set@font@type: Added 'MM' font type; tests true, e.g., with Skia & Minion MM. Used with the OpticalSize feature.	34
Removed exclusivity from font type (AAT, OpenType) check, since fonts can be both.	34
Removed various \count255s.	34
\zf@update@ff: Fix for featureless fonts (e.g., the MS fonts) being ignored.	37
v1.8	
\AtBeginDocument: Added support for user-specified \mathrm and others.	54

Finally fixed legacy maths font issues. Also checks that euler.sty is loaded in the right order.	54
\setmathrm: Implemented (with friends).	28
v1.8a	
\AtBeginDocument: Added conditional to \colon math symbol (incompatibility with lucida and amsmath)	54
v1.9	
General: CharacterShape now CJKShape	48
SMALLCAPS option changed to UppercaseSmallCaps to facilitate option normalisation (to come). Similarly for PETITECAPS.	46
Swashes feature changed to Contextuals. Option of this feature Contextual changed to Swash, for obvious reasons.	46
TextSpacing now CharacterWidth, with associated option names' change.	48
Alternate/Variant options can be assigned names.	47
New Scale options: MatchLowercase and MatchUppercase.	42
New feature HyphenChar.	44
New feature Kerning.	47
New feature PunctuationSpace.	43
New feature UprightFeatures.	40
New feature Vertical.	48
New feature WordSpace.	43
New features SmallCapsFont and SmallCapsFeatures.	40
Package options (no)config, quiet implemented.	57
\addfontfeatures: Added \ignorespaces to make it invisible.	29
Changed \fontspec call to \@fontspec so that \ignorespaces isn't called unnecessarily.	29
\aliasfontfeature: Implemented.	30
\aliasfontfeatureoption: Implemented.	30
\AtBeginDocument: Maths hex numbers converted to decimal.	54
Suppresses harmless maths font encoding warnings.	57
\emph: Redefined \em in order for nested emphases to work.	53
\fontspec: Added \ignorespaces to make it invisible.	27
\keyval@alias@key: Implemented.	38
\multi@alias@key: Implemented for \aliasfontfeature.	38
\newAATfeature: Replacement for \newfeaturecode.	30
\newfontlanguage: Implemented.	31
\newfontscript: Implemented.	31
\newICUfeature: Implemented.	30
\zf@calc@scale: Implemented for auto-scaling options.	43
\zf@check@ot@feat: Implemented.	40
\zf@check@ot@lang: Implemented.	40
\zf@check@ot@script: Implemented.	39
\zf@DeclareFontShape: Implemented as wrapper for \DeclareFontShape.	35
Slanted/italic shape substitution implemented.	35
\zf@fontspec: Absorbed the comma into \zf@. .@options as to be more efficient when they are not defined.	34
Abstracted the long family name so the NFSS family is simple.	34
Incorporated \zf@get@feature@requests argument change.	34
Incorporated \zf@make@font@shapes change; removed \zf@options storage macro.	34



\zf@get@feature@requests: Absorbed comma into \zf@default@options,	36
making \zf@current@options redundant.	36
Added an argument to eliminate the \zf@options macro.	36
Removed init stuff.	36
\zf@init: Taken from \zf@get@feature@requests.	36
\zf@make@feature: Now checks for OpenType feature.	37
\zf@make@font@shapes: \zf@scale@str eliminated.	35
Absorbed \IfEqFonts.	35
Added argument for \zf@get@feature@requests change.	35
Added code for SmallCaps . . . features.	35
Added logging of /B, /I, /BI failure.	35
Changed input syntax.	35
Incorporated \sdefault test into the \DeclareFontShape argument di-	35
rectly now that it's fully expanded.	35
Made local to hide \zf@fontname changes.	35
Removed \zf@scshape macro.	35
Removed \nfss@catcodes wrapper.	35
\zf@make@smallcaps: Now uses \zf@check@ot@feat.	37
\zf@partial@fontname: Implemented.	41
\zf@update@family: Now fully expands arguments.	36
\zf@update@ff: Removed ridiculous \zf@feature@separator code.	37
\zf@v@strnum: Implemented.	39
\zf@wordspace@parse: Implemented.	43

## 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	
\!	48
\,	48
\@	15
\empty	83, 86, 154, 173, 186, 193, 203–205, 235, 247, 249, 282–284, 288–299, 306, 309, 326, 363, 364, 367, 368, 385, 392, 394, 436, 446, 460, 466, 488, 534, 562, 696, 701
\font@info	968
\font@warning	967, 968
\gobble	397
\ifnextchar	67, 75
\ifpackageloaded	964, 969, 979–981
\nameuse	176
\ne	407, 419, 431
\nil	383–386, 397, 440, 450, 455, 464, 468
\onlypreamble	59–62
\tempa	70, 73, 78, 81, 88, 92, 234, 235, 237, 241, 266, 269–272, 275, 285, 286, 435, 436, 441, 445, 446, 451, 456, 459, 460, 465, 470, 472, 504, 506, 510, 533, 534, 556, 558, 702, 944, 948, 951, 995, 997
\tempb	240, 241, 264, 265, 268, 270, 271, 505, 506, 509, 510, 557, 558, 562, 582, 945, 946
\tempcnta	123, 134, 387, 389, 391, 393, 395, 403, 415, 427, 922, 929
\tempcntb	388–395, 400, 402, 405, 412, 414, 417, 423, 426, 429
\tempdima	521, 523, 535–537, 539, 544, 549, 550
\tempdimb	522, 523, 536, 540, 545
\tempdimc	523, 524, 537, 541, 546
\tempfonta	236, 237
\tempfontb	239, 240, 245
\tempswafalse	200, 201, 401, 413, 425
\tempswatrue	199, 404, 416, 428
\[	47
\]	24, 25, 42
\]	49
A	
\acute	984
\addfontfeature	7, 96
\addfontfeatures	7, <u>84</u>
\advance	389, 391, 393, 395, 407, 419, 431
\aliasfontfeature	10–15, 25, <u>117</u>
\aliasfontfeatureoption	16–18, 25, <u>117</u>
\AtBeginDocument	<u>964</u>
B	
\bar	988
\begin	19, 28, 41
\begingroup	146, 519, 994
\bfdefault	187, 189, 206, 208, 211, 214, 1042, 1045, 1048, 1051, 1052, 1054, 1055
\bfseries	43
\bgroup	233
\breve	989
C	
\c@zf@index	16, 401–403, 405, 407, 413–415, 417, 419, 425–427, 429, 431
\c@zf@language	18, 134, 302, 423, 427, 922, 929
\c@zf@newff	15, 99
\c@zf@script	17, 123, 300, 412, 415, 423, 427
\check	990
\colon	997, 998
\cos	46
\csgname	90, 91, 101, 142–144, 174
\cyrillicencoding	26, 29
D	
\ddot	986
\DeclareFontFamily	183
\DeclareFontShape	267, 273
\DeclareMathAccent	984–993
\DeclareMathDelimiter	1030–1034
\DeclareMathSymbol	998, 1002–1005, 1007–1029, 1035
\DeclareOption	1057, 1061, 1062
\DeclareRobustCommand	12, 71, 79, 939, 952, 955, 958, 961
\DeclareSymbolFont	982, 1038
\DeclareTextFontCommand	942

<code>\def</code> . . . . .	24, 63–65, 68, 76, 82, 156, 159, 264, 301, 303, 384, 386, 468, 475, 478, 481, 484, 487, 532, 533, 582
<code>\def@cx</code> . . . . .	99, <u>142</u> , 172
<code>\defaultfontfeatures</code> . . . . .	6, 6, 51, <u>82</u>
<code>\define@choicekey</code> . . . . .	489
<code>\define@key</code> . . . . .	8, 100, 120, 131, 348, 350, 434, 444, 454, 458, 474, 477, 480, 483, 486, 493, 498, 503, 527, 547, 551, 554, 583, 587, 590, 593, 596, 694, 699, 748, 919
<code>\Delta</code> . . . . .	1018
<code>\document</code> . . . . .	28
<code>\documentclass</code> . . . . .	2
<code>\dot</code> . . . . .	992
<code>\dots</code> . . . . .	42, 43
<b>E</b>	
<code>\edef</code> . . . . .	70, 78, 87, 88, 142, 148, 152, 234, 237, 240, 244, 255, 266, 270, 272, 285, 310, 316, 362, 366, 370, 435, 438, 442, 445, 448, 452, 457, 459, 462, 470, 472, 476, 479, 482, 485, 490, 492, 494, 496, 499, 501, 504, 505, 509, 513, 517, 543, 550, 556, 557, 598, 702, 944, 945, 948
<code>\egroup</code> . . . . .	261, 996
<code>\else</code> . . . . .	127, 138, 157, 169, 188, 195, 207, 210, 213, 235, 243, 249, 251, 253, 257, 321, 330, 340, 365, 369, 373, 376, 392, 394, 406, 418, 430, 439, 449, 463, 471, 508, 512, 538, 560, 567, 572, 575, 605, 606, 751, 926, 933, 972, 1001, 1006, 1050
<code>\em</code> . . . . .	<u>964</u>
<code>\emph</code> . . . . .	<u>964</u>
<code>\encodingdefault</code> . . . . .	25
<code>\end</code> . . . . .	37, 44, 55
<code>\endcsname</code> . . . . .	90, 91, 101, 104, 111, 142–144, 167, 168, 174, 177, 947
<code>\endgroup</code> . . . . .	218, 526, 1000
<code>\ExecuteOptions</code> . . . . .	1063
<code>\expandafter</code> . . . . .	142–144, 169, 170, 174, 392, 394, 397, 563, 565, 566
<b>F</b>	
<code>\f@encoding</code> . . . . .	947
<code>\f@family</code> . . . . .	90, 91, 947
<code>\f@series</code> . . . . .	947
<code>\f@shape</code> . . . . .	946
<code>\f@size</code> . . . . .	149, 162, 236, 239, 491, 495, 500
<code>\fi</code> . . . . .	106, 113, 129, 141, 160, 161, 163, 170, 175, 190, 191, 197, 198, 200, 201, 209, 212, 215–217, 228, 231, 238, 251, 252, 257–260, 276, 280, 311, 312, 317, 318, 321, 322, 333, 334, 345, 346, 378–381, 392, 394, 408, 420, 432, 443, 453, 467, 473, 487, 514, 515, 531, 542, 571, 579–581, 600, 604, 610–612, 698, 703, 754, 936, 937, 949, 950, 978, 999, 1036, 1037, 1053
<code>\font</code> . . . . .	149, 162, 236, 239, 265, 491, 495, 500, 521, 544–546, 550, 559, 566, 574
<code>\font@warning</code> . . . . .	1056
<code>\fontdimen</code> . . . . .	521, 522, 535, 539–541, 544–546, 549, 550
<code>\fontfamily</code> . . . . .	33, 72, 80, 93
<code>\fontname</code> . . . . .	128, 139, 237, 240, 329, 343, 569, 577, 609, 934
<code>\fontshape</code> . . . . .	941, 951
<code>\fontspec</code> . . . . .	3, 25, 30, <u>31</u> , 39
<b>G</b>	
<code>\g@addto@macro</code> . . . . .	28, 559, 564, 574
<code>\Gamma</code> . . . . .	1017
<code>\gdef@cx</code> . . . . .	<u>142</u> , 180–182
<code>\global</code> . . . . .	123, 134
<code>\grave</code> . . . . .	985
<b>H</b>	
<code>\hat</code> . . . . .	991
<code>\hbox</code> . . . . .	14, 15
<code>\hyphenchar</code> . . . . .	559, 565, 574
<b>I</b>	
<code>\if</code> . . . . .	469
<code>\if@tempswa</code> . . . . .	122, 133, 202, 315, 337, 921, 928
<code>\ifcase</code> . . . . .	222
<code>\ifcsname</code> . . . . .	104, 111, 167, 168, 177, 947
<code>\ifdefined</code> . . . . .	1046
<code>\ifnum</code> . . . . .	226, 371, 402, 403, 414, 415, 426, 427, 563, 573
<code>\ifodd</code> . . . . .	374
<code>\ifx</code> . . . . .	154, 186, 193, 203–205, 235, 241, 247, 249, 251, 257, 271, 309, 326, 363, 367, 392, 394, 436, 446, 460, 506, 510, 534, 558, 562, 696, 701, 946, 997
<code>\ifzf@atsui</code> . . . . .	8, 155, 307, 324
<code>\ifzf@euler@package@loaded</code> . . . . .	13, 970
<code>\ifzf@firsttime</code> . . . . .	3, 278, 320, 487, 529, 607
<code>\ifzf@icu</code> . . . . .	9, 158, 313, 321, 335, 597, 605, 749



<b>R</b>		<b>U</b>	
<code>\ratio</code> .....	523	<code>\unless</code> .....	104, 111, 167, 177, 185, 192, 254, 309, 320, 487, 529, 696, 701
<code>\reflectbox</code> .....	14	<code>\updefault</code> .....	184, 187, 189, 963, 1038–1040, 1042–1045, 1047, 1048, 1051, 1054, 1055
<code>\relax</code> ...	123, 134, 300, 302, 388, 469, 559, 574, 922, 929, 940, 956, 959, 962	<code>\upshape</code> .....	<u>952</u>
<code>\repeat</code> .....	409, 421, 433	<code>\Upsilon</code> .....	1024
<code>\RequirePackage</code> .....	1, 22, 23	<code>\usepackage</code> .....	4
<code>\RequireXeTeX</code> .....	2		
<code>\rmdefault</code> .....	37, 63	<b>V</b>	
<code>\rmfamily</code> .....	520	<code>\verb</code> .....	25, 39, 51
<b>S</b>		<b>X</b>	
<code>\scdefault</code> ..	251, 257, 954, 957, 960, 963	<code>\xdef</code> .....	124–126, 135–137, 143, 176, 279, 321, 524, 923–925, 930–932
<code>\scshape</code> .....	42, <u>952</u>	<code>\XeTeX</code> .....	11, 12, 24, 46
<code>\section</code> .....	22	<code>\XeTeXcharglyph</code> .....	563, 573
<code>\selectfont</code> .....	33, 72, 80, 93, 941, 951	<code>\XeTeXcountvariations</code> .....	226
<code>\setboldmathrm</code> .....	6, <u>47</u>	<code>\XeTeXfeaturename</code> .....	362
<code>\setkeys</code> ..	151, 285, 348, 497, 502, 695, 700	<code>\XeTeXfonttype</code> .....	222
<code>\setlength</code> ..	521–523, 535, 539–541, 549	<code>\XeTeXisexclusivefeature</code> .....	371
<code>\SetMathAlphabet</code> .....	1040–1044, 1047–1049, 1051, 1052, 1054, 1055	<code>\XeTeXOTcountfeatures</code> .....	423
<code>\setmathrm</code> .....	6, <u>47</u>	<code>\XeTeXOTcountlanguages</code> .....	412
<code>\setmathsf</code> .....	6, <u>47</u>	<code>\XeTeXOTcountscripts</code> .....	400
<code>\setmathtt</code> .....	6, <u>47</u>	<code>\XeTeXOTfeaturetag</code> .....	427
<code>\setmonofont</code> .....	4, 9, <u>35</u> , 39	<code>\XeTeXOTlanguagetag</code> .....	415
<code>\setromanfont</code> .....	4, 7, <u>35</u> , 39	<code>\XeTeXOTscripttag</code> .....	403
<code>\setsansfont</code> .....	4, 8, <u>35</u> , 39	<code>\XeTeXselectorname</code> .....	366
<code>\SetSymbolFont</code> .....	983, 1039, 1045	<code>\Xi</code> .....	1021
<code>\sfdefault</code> .....	41, 64	<code>\XKV@rm</code> .....	152, 696, 701
<code>\sffamily</code> .....	43	<code>\XKV@tfam</code> .....	328, 342
<code>\sdefault</code> .....		<code>\XKV@tkey</code> .....	328, 342
	251, 257, 938, 941, 954, 957, 960, 963	<b>Z</b>	
<code>\Sigma</code> .....	1023	<code>\z@</code> .....	387, 401, 413, 425
<code>\sishape</code> .....	<u>938</u>	<code>\zap@space</code> .....	173, 466, 488
<code>\sldefault</code> .....	273, 957	<code>\zf@@</code> .....	530, 532, 561, 582
<code>\slshape</code> .....	955, 956	<code>\zf@ii</code> .....	530
<code>\smash</code> .....	13	<code>\zf@iii</code> .....	530
<code>\space</code> .....	265, 525, 569, 577, 609	<code>\zf@adjust</code> .....	
<code>\stepcounter</code> .....	98, 169		268, 274, 284, 543, 550, 559, 564, 574
<code>\string</code> .....	265	<code>\zf@atsuifalse</code> .....	221
<code>\strip@pt</code> .....	524	<code>\zf@atsuitrue</code> .....	225
<b>T</b>		<code>\zf@basefont</code> ....	128, 139, 149, 162, 222, 226, 245, 329, 343, 362, 366, 371, 400, 403, 412, 415, 423, 427, 491, 495, 500, 522, 535, 539–541, 549, 563, 569, 573, 577, 609, 934
<code>\TeX</code> .....	51	<code>\zf@bf</code> .....	186, 189, 204, 211, 291, 441
<code>\textsf</code> .....	22, 24, 46	<code>\zf@bf@feat</code> .....	187, 189, 296, 478
<code>\textsi</code> .....	<u>938</u>	<code>\zf@bfit</code> .....	203, 214, 293, 456
<code>\the</code> .....	99, 174, 544–546, 550	<code>\zf@bfit@feat</code>	206, 208, 211, 214, 298, 484
<code>\Theta</code> .....	1019		
<code>\tilde</code> .....	987		
<code>\ttdefault</code> .....	45, 65		
<code>\two@digits</code> .....	702		
<code>\typeout</code> .....	1059, 1060		

\zf@calc@scale . . . . .	507, 511, <u>518</u>	\zf@make@feature 8, <u>323</u> , 350, 697, 702, 750
\zf@check@one@char . . . . .	561, 582	\zf@make@font@shapes . . . . .
\zf@check@ot@feat . . . . .	314, 336, <u>422</u>	184, 187,
\zf@check@ot@lang . . . . .	132, <u>410</u> , 920, 927	189, 194, 196, 206, 208, 211, 214, <u>232</u>
\zf@check@ot@script . . . . .	121, <u>398</u>	\zf@make@smallcaps . . . . .
\zf@DeclareFontShape . . . . .	246, 250, 256, <u>262</u>	248, <u>305</u>
\zf@default@options . . . . .		\zf@math@eulertrue . . . . .
. . . . .	82, 83, 85, 86, 94, 179, 181, 285	971
\zf@default@options@old . . . . .	85, 94	\zf@math@lucidatrue . . . . .
\zf@define@feature@option . . . . .		979–981
109, 116, <u>347</u> , 614–637, 639–647,		\zf@merge@shape . . . . .
649–656, 658–669, 671–673,		<u>943</u> , 954, 957, 960, 963
675–680, 682–688, 690–693,		\zf@mmfalse . . . . .
705–715, 717–723, 725–732, 734–746		221
\zf@define@font@feature . . . . .	105,	\zf@mmtrue . . . . .
112, <u>347</u> , 613, 638, 648, 657, 670,		227
674, 681, 689, 704, 716, 724, 733, 747		\zf@nobftrue . . . . .
\zf@enc . . . . .	24–27,	437
29, 30, 183, 267, 273, 1038–1045,		\zf@noittrue . . . . .
1047–1049, 1051, 1052, 1054, 1055		447
\zf@euler@package@loadedfalse . . . . .	965	\zf@nosctrue . . . . .
\zf@euler@package@loadedtrue . . . . .	964	461
\zf@family . . . . .	33,	\zf@PackageError . . . . .
37, 41, 45, 49, 52, 55, 58, 72, 80,		19, 357, 568, 576, 973
93, 176, 177, 180–183, 267, 273, 274		\zf@PackageInfo . . . . .
\zf@family@long . . . . .		21, 178, 242, 265, 525, 1062
125, 136, 150, 167, 172, 176, 279,		\zf@PackageWarning . . . . .
438, 442, 448, 452, 457, 462, 476,		20, 108,
479, 482, 485, 492, 496, 501, 924, 931		115, 128, 139, 327, 341, 608, 934, 1062
\zf@ff . . . . .	182, 264, 282, 321	\zf@partial@fontname . . . . .
\zf@firsttimefalse . . . . .	166	440, 450, 455, 464, <u>468</u>
\zf@firsttimetrue . . . . .	164	\zf@pre@ff 126, 137, 182, 264, 288, 925, 932
\zf@font@feat . . . . .	152, 165, 184, 187,	\zf@rmboldmaths . . . . .
189, 194, 196, 206, 208, 211, 214, 289		52, 1046–1049
\zf@font@warning . . . . .	967, 1056	\zf@rmmaths . . . . .
\zf@fontname . . . . .	148–150,	49, 63, 1038–1042, 1045, 1051, 1052
162, 180, 182, 184, 187, 194, 206,		\zf@sc . . . . .
244, 255, 264, 470, 491, 495, 500, 525		247, 255, 294, 465
\zf@fontspec . . . . .	32, 36,	\zf@sc@feat . . . . .
40, 44, 48, 51, 54, 57, 69, 77, 89, <u>145</u>		251, 257, 299, 487
\zf@get@feature@requests 165, 263, <u>281</u>		\zf@scale 268, 283, 513, 516, 517, 524, 525
\zf@icufalse . . . . .	221	\zf@script@name . . . . .
\zf@icutrue . . . . .	230	124, 140, 301, 344, 935
\zf@init . . . . .	147, <u>287</u>	\zf@set@font@type . . . . .
\zf@it . . . . .	193, 196, 205, 208, 292, 451	153, <u>220</u>
\zf@it@feat . . . . .	194, 196, 297, 481	\zf@sfmaths . . . . .
\zf@iv@strnum . . . . .	<u>382</u> , 399, 411	55, 64, 1043, 1054
\zf@iv@strnum@i . . . . .	383, 384, 397	\zf@smallcaps . . . . .
\zf@iv@strnum@ii . . . . .	385, 386	249, 250, 306, 310, 316
\zf@language@name 135, 303, 344, 923, 930		\zf@suffix . . . . .
\zf@make@aat@feature@string . . . . .		154,
. . . . .	308, 325, <u>361</u>	156, 159, 162, 182, 236, 239, 264,
		290, 490, 491, 494, 495, 499, 500, 598
		\zf@tfm . . . . .
		223
		\zf@tfmfalse . . . . .
		221
		\zf@this@featurename . . . . .
		362, 363, 372, 375, 377
		\zf@this@selectionname . . . . .
		366, 367, 372, 375, 377
		\zf@thisfontfeature . . . . .
		309, 310, 326, 332, 364, 368, 370
		\zf@thisinfo . . . . .
		87
		\zf@ttmaths . . . . .
		58, 65, 1044, 1055
		\zf@up@feat . . . . .
		184, 295, 475
		\zf@update@family . . . . .
		101, <u>277</u> , 331,
		338, 466, 488, 516, 528, 548, 552,
		555, 584, 588, 591, 594, 599, 602, 752
		\zf@update@ff . . . . .
		102, <u>319</u> , 332,
		339, 553, 585, 589, 592, 595, 603, 753
		\zf@v@strnum . . . . .
		<u>382</u> , 424
		\zf@wordspace@parse . . . . .
		530, <u>532</u>