

The fontspec package

WILL ROBERTSON and KHALED HOSNY

2010/05/27 v2.0b1

Contents

| | | | |
|--|-----------|--|-----------|
| 1 History | 2 | 7.5 Different features for different font sizes | 13 |
| 2 Introduction | 3 | 8 Font independent options | 14 |
| 2.1 About this manual | 3 | 8.1 Color | 14 |
| 3 Package loading and options | 3 | 8.2 Scale | 15 |
| 3.1 Maths fonts adjustments | 4 | 8.3 Interword space | 15 |
| 3.2 Configuration | 4 | 8.4 Post-punctuation space | 16 |
| 3.3 Warnings | 4 | 8.5 The hyphenation character | 16 |
| | | 8.6 Optical font sizes | 16 |
| I General font selection | 6 | II OpenType | 18 |
| 4 Font selection | 6 | 9 Introduction | 18 |
| 4.1 By font name | 6 | 10 Complete listing of OpenType font features | 18 |
| 4.2 By file name | 6 | 10.1 Ligatures | 18 |
| 5 Default font families | 8 | 10.2 Letters | 19 |
| 6 New commands to select font families | 8 | 10.3 Numbers | 19 |
| 6.1 More control over font shape selection | 9 | 10.4 Contextuals | 20 |
| 6.2 Math(s) fonts | 10 | 10.5 Vertical Position | 20 |
| 6.3 Miscellaneous font selecting details | 11 | 10.6 Fractions | 21 |
| 7 Selecting font features | 11 | 10.7 StylisticSet | 22 |
| 7.1 Default settings | 11 | 10.8 Alternates | 22 |
| 7.2 Changing the currently selected features | 12 | 10.9 Style | 23 |
| 7.3 Priority of feature selection | 12 | 10.10Diacritics | 24 |
| 7.4 Different features for different font shapes | 12 | 10.11Kerning | 24 |
| | | 10.12CJK shape | 24 |
| | | 10.13Character width | 25 |
| | | 10.14Vertical typesetting | 26 |
| | | 10.15OpenType scripts and languages | 26 |

| | | | |
|---|-----------|---|------------|
| III Fonts and features with X_YTeX | 29 | V The patching/improvement of L^AT_EX₂_ε and other packages | 40 |
| 11 X _Y TeX-only font features | 29 | 18 Inner emphasis | 40 |
| 11.1 Mapping | 29 | 19 Unicode footnote symbols | 40 |
| 11.2 Letter spacing | 29 | 20 Verbatim | 40 |
| 11.3 Font transformations | 29 | 21 Discretionary hyphenation: \- 40 | |
| 11.4 Different font technologies: AAT and ICU | 30 | | |
| 11.5 Optical font sizes | 30 | | |
| 12 Mac OS X's AAT fonts | 30 | VI fontspec.sty | 41 |
| 12.1 Ligatures | 31 | 22 Implementation | 41 |
| 12.2 Letters | 31 | 22.1 Bits and pieces | 41 |
| 12.3 Numbers | 31 | 22.2 Error/warning messages | 42 |
| 12.4 Contextuals | 31 | 22.3 Option processing | 45 |
| 12.5 Vertical position | 32 | 22.4 Packages | 45 |
| 12.6 Fractions | 32 | 22.5 Encodings | 45 |
| 12.7 Variants | 33 | 22.6 User commands | 46 |
| 12.8 Alternates | 33 | 22.7 Programmer's interface | 51 |
| 12.9 Style | 33 | 22.8 Internal macros | 54 |
| 12.10CJK shape | 34 | 22.9 keyval definitions | 69 |
| 12.11Character width | 34 | 22.10Italic small caps | 86 |
| 12.12Annotation | 35 | 22.11Selecting maths fonts | 87 |
| 12.13Vertical typesetting | 35 | 22.12Finishing up | 91 |
| 12.14Diacritics | 35 | | |
| 12.15Annotation | 36 | | |
| 13 AAT & Multiple Master font axes | 36 | VII fontspec.lua | 92 |
| IV Programming interface | 37 | VIII fontspec-patches.sty | 96 |
| 14 Defining new features | 37 | 22.13Unicode footnote symbols | 96 |
| 15 Going behind fontspec's back | 38 | 22.14Emph | 96 |
| 16 Renaming existing features & options | 38 | 22.15\- | 96 |
| 17 Programming details | 38 | 22.16Verbatims | 96 |
| | | IX fontspec.cfg | 99 |
| | | X Example documents | 100 |

1 History

This package began life as a L^AT_EX interface to select system-installed Mac OS X fonts in Jonathan Kew's X_YTeX, the first widely-used unicode extension to T_EX.

Over time, X_YTeX was extended to support OpenType fonts and then was ported into a cross-platform program to run also on Windows and Linux.

More recently, LuaTeX is fast becoming the TeX engine of the day; it supports unicode encodings and OpenType fonts and opens up the internals of TeX via the Lua programming language. Hans Hagen's ConTeXt Mk. IV is a re-write of his powerful typesetting system, taking full advantage of LuaTeX's features including font support; a kernel of his work in this area has been extracted to be useful for other TeX macro systems as well, and this has enabled fontspec to be adapted for L^ATeX when run with the LuaTeX engine. Elie Roux and Khaled Hosny have been instrumental and invaluable with this development work.

2 Introduction

The fontspec package allows users of either X_YTeX or LuaTeX to load OpenType fonts in a L^ATeX document. No font installation is necessary, and font features can be selected and used as desired throughout the document.

Without fontspec, it is necessary to write cumbersome font definition files for L^ATeX, since L^ATeX's font selection scheme (known as the 'NFSS') has a lot going on behind the scenes to allow easy commands like `\emph` or `\bfseries`. With an uncountable number of fonts now available for use, however, it becomes less desirable to have to write these font definition (`.fd`) files for every font one wishes to use.

Because fontspec is designed to work in a variety of modes, this user documentation is split into separate sections that are designed to be relatively independent. Nonetheless, the basic functionality all behaves in the same way, so previous users of fontspec under X_YTeX should have little or no difficulty switching over to LuaTeX.

This manual can get rather in-depth, as there are a lot of details to cover. See the example documents `fontspec-xetex.tex` and `fontspec-luatex.tex` for a complete minimal example with each engine.

2.1 About this manual

As of version 2.0 with the addition of LuaTeX support, this manual is still in the process of being re-written. It currently contains no typeset examples. Your patience and feedback is appreciated!

3 Package loading and options

For basic use, no package options are required:

```
\usepackage{fontspec}
```

In brief, fontspec's package options are
`no-math` Don't change any maths fonts;
`no-config` Don't load `fontspec.cfg`; and,
`quiet` Output fontspec warnings in the log file rather than the console output.
These are described in more detail below.

X_YTeX users only Ross Moore’s xunicode package is recommended for providing backwards compatibility with L^AT_EX’s 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^AT_EX under X_YTeX; it also provides the `\XeTeX` macro to typeset the X_YTeX logo.

LuaTeX users only In order to load fonts by their name rather than by their filename (e.g., ‘Latin Modern Roman’ instead of ‘ec-lmr10’), you may need to run the script `mkluatexfontdb.lua`, which is distributed with the luaotfload package. Please see its documentation for more information. Note that the xunicode package is not required as it has been incorporated directly into the unicode font definitions (see the euenc package for more information).

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

3.1 Maths fonts adjustments

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

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

3.2 Configuration

If you wish to customise any part of the fontspec interface (see later in this manual, [Section 14 on page 37](#) and [Section 16](#)), this should be done by creating your own `fontspec.cfg` file,¹ which will be automatically loaded if it is found by X_YTeX. Either place it in the same folder as the main document for isolated cases, or in a location that X_YTeX or LuaTeX searches by default; e.g. in MacTeX: `~/Library/texmf/tex/latex/`. The package option `[no-config]` will suppress this behaviour under all circumstances.

3.3 Warnings

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

¹An example is distributed with the package.

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

Part I

General font selection

This section concerns the variety of commands that can be used to select fonts.

```
\fontspec [<font features>] {<font name>}  
\setmainfont [<font features>] {<font name>}  
\setsansfont [<font features>] {<font name>}  
\setmonofont [<font features>] {<font name>}  
\newfontfamily <cmd> [<font features>] {<font name>}
```

These are the main font-selecting commands of this package. The `\fontspec` command selects a font for one-time use; all others should be used to define the standard fonts used in a document. They will be described later in this section.

The font features argument accepts comma separated *=<option>* lists; these are described in later:

- For general font features, see [Section 8 on page 14](#)
- For OpenType fonts, see [Part II on page 18](#)
- For XeTeX-only general font features, see [Part III on page 29](#)
- For features for AAT fonts in XeTeX, see [Part Section 12 on page 30](#)

4 Font selection

In both LuaTeX and XeTeX, fonts can be selected either by ‘font name’ or by ‘file name’.

4.1 By font name

Fonts known to LuaTeX or XeTeX may be loaded by their names. ‘Known to’ in this case generally means ‘exists in a standard fonts location’ such as `~/Library/Fonts` on Mac OS X, or `C://WINNT/Fonts` on Windows.

The simplest example might be something like

```
\fontspec[ ... ]{Cambria}
```

in which the bold and italic fonts will be found automatically (if they exist) and are immediately accessible with the usual `\textit` and `\textbf` commands.

TODO: add explanation for how to find out what the ‘font name’ is.

4.2 By file name

When selecting fonts by file name, any font that can be found in the default search paths may be used directly (including in the current directory) without having to explicitly define the location of the font file on disk.

Note that XeTeX with the `xdvipdfmx` driver and LuaTeX can both select fonts in this way, but XeTeX with the `xdv2pdf` driver can only select fonts by name and not

by file name. The xdvipdfmx driver is default for X_YTeX; the xdv2pdf driver is only available on Mac OS X.

Fonts selected by filename must include bold and italic variants explicitly.

```
\fontspec
[ BoldFont      = texgyrepagella-bold.otf ,
  ItalicFont    = texgyrepagella-italic.otf ,
  BoldItalicFont = texgyrepagella-bolditalic.otf ]
{texgyrepagella-regular.otf}
```

fontspec knows that the font is to be selected by file name by the presence of the ‘.otf’ extension. An alternative is to specify the extension separately, as shown following:

```
\fontspec
[ Extension      = .otf ,
  BoldFont       = texgyrepagella-bold ,
  ... ]
{texgyrepagella-regular}
```

If desired, an abbreviation can be applied to the font names based on the mandatory ‘font name’ argument:

```
\fontspec
[ Extension      = .otf ,
  UprightFont    = *-regular ,
  BoldFont       = *-bold ,
  ... ]
{texgyrepagella}
```

In this case ‘texgyrepagella’ is no longer the name of an actual font, but is used to construct the font names for each shape; the * is replaced by ‘texgyrepagella’. Note in this case that UprightFont is required for constructing the font name of the normal font to use.

To load a font that is not in one of the default search paths, its location in the filesystem must be specified with the Path feature:

```
\fontspec
[ Path           = /Users/will/Fonts/ ,
  UprightFont    = *-regular ,
  BoldFont       = *-bold ,
  ... ]
{texgyrepagella}
```

Note that X_YTeX and Lua_{TeX} are able to load the font without giving an extension, but fontspec must know to search for the file; this can be indicated by declaring the font exists in an ‘ExternalLocation’:

```
\fontspec
[ ExternalLocation ,
  BoldFont         = texgyrepagella-bold ,
  ... ]
{texgyrepagella-regular}
```

To be honest, Path and ExternalLocation are actually the same feature with different names. The former can be given without an argument and the latter can be given with one; the different names are just for clarity.

5 Default font families

```
\setmainfont [<font features>] {<font name>}
\setsansfont [<font features>] {<font name>}
\setmonofont [<font features>] {<font name>}
```

These commands are used to select the default font families for the entire document. They take the same arguments as \fontspec. For example:

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

Here, the scales of the fonts have been chosen to equalise their lowercase letter heights. The Scale font feature will be discussed further in [Section 8 on page 14](#), including methods for automatic scaling.

6 New commands to select font families

```
\newfontfamily \<font-switch> [<font features>] {<font name>}
\newfontface \<font-switch> [<font features>] {<font name>}
```

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.

`\newfontfamily` For this reason, new commands can be created for loading a particular font family with the \newfontfamily command:

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

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

`\newfontface` Sometimes only a specific font face is desired, without accompanying italic or bold variants begin automatically selected. 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
```

This example is repeated in [Section 12.4 on page 31](#).

6.1 More control over font shape selection

| |
|---|
| <pre>BoldFont = ItalicFont = BoldItalicFont = SlantedFont = BoldSlantedFont = SmallCapsFont = </pre> |
|---|

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

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

If a bold italic shape is not defined, or you want to specify *both* custom bold and italic shapes, the `BoldItalicFont` feature is provided.

6.1.1 Input shorthands

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

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

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

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

6.1.2 Small caps and slanted font shapes

For the rare situations where a font family will have slanted *and* italic shapes, these may be specified separately using the analogous features `SlantedFont` and `BoldSlantedFont`. Without these, however, the \LaTeX font switches for slanted (`\textsl`, `\slshape`) will default to the italic shape.

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}
```

All of the bold, italic, and small caps fonts can be loaded with different font features from the main font. See ?? for details.

6.2 Math(s) fonts

When `\setmainfont`, `\setsansfont` and `\setmonofont` are used in the preamble, they also define the fonts to be used in maths mode inside the `\mathrm`-type commands. This only occurs in the preamble because \TeX 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.²)

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

```
\setmathrmfont [⟨font features⟩] {⟨font name⟩}
\setmathsffont [⟨font features⟩] {⟨font name⟩}
\setmathttf font [⟨font features⟩] {⟨font name⟩}
\setboldmathrmfont [⟨font features⟩] {⟨font name⟩}
```

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

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

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

²Speaking of `euler`, if you want to use its `[mathbf]` option, it won't work, and you'll need to put this after `fontspec` is loaded instead: `\AtBeginDocument{\DeclareMathAlphabet\mathbf{U}{eur}{b}{n}}`

and this would allow you to typeset something like this:

```
\setmainfont{Optima Regular}
$ X \rightarrow \mathrm{X} \rightarrow \mathbf{X} $
\\boldmath
$ X \rightarrow \mathrm{X} \rightarrow \mathbf{X} $
```

(TODO: this example obviously doesn't work without the graphic.)

6.3 Miscellaneous font selecting details

Spaces `\fontspec` and `\addfontfeatures` ignore trailing spaces as if it were a 'naked' control sequence; e.g., 'M. `\fontspec{...}` N' and 'M. `\fontspec{...}N`' are the same.

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

Emphasis and nested emphasis You may specify the behaviour of the `\emph` command by setting the `\emshape` command. E.g., for bold emphasis:

```
\renewcommand\emshape{\bfseries}
```

Nested emphasis is controlled by the `\eminnershape` command. For example, for `\emph{\emph{...}}` to produce small caps:

```
\renewcommand\eminnershape{\scshape}
```

X_YTeX users will need to load the `xltxtra` package before the advice above works.

7 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^AT_EX.

7.1 Default settings

`\defaultfontfeatures{}`

It is desirable to define options that are applied to every subsequent font selection command: a default feature set, so to speak. This may be defined with the `\defaultfontfeatures` command. New calls of `\defaultfontfeatures` overwrite previous ones.

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

7.2 Changing the currently selected features

`\addfontfeatures{}`

This 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:

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

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

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

7.3 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.

7.4 Different features for different font shapes

```
BoldFeatures{<features>}
ItalicFeatures{<features>}
BoldItalicFeatures{<features>}
SlantedFeatures{<features>}
BoldSlantedFeatures{<features>}
SmallCapsFeatures{<features>}
```

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.

```

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

```

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

```

\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, features specified with SmallCapsFeatures 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.

```

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

```

7.5 Different features for different font sizes

```

SizeFeatures = {
  ...
  { Size = <size range>, <font features> }
  { Size = <size range>, Font = <font name>, <font features> }
  ...
}

```

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

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

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

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

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

A less trivial example is shown in the context of optical font sizes in [Section 11.5 on page 30](#).

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

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

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

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

8 Font independent options

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

8.1 Color

Color (or Colour), also shown in [Section 7.1 on page 11](#) and elsewhere, uses X_YTeX font specifications to set the color of the text. The color 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}
```

8.2 Scale

```
Scale = <number>
Scale = MatchLowercase
Scale = MatchUppercase
```

In its explicit form, `Scale` takes a single numeric argument for linearly scaling the font, as demonstrated in [Section 5 on page 8](#). It is now possible to measure the correct dimensions of the fonts loaded and calculate values to scale them automatically.

As well as a numerical argument, `Scale` feature also accepts 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.

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

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

8.3 Interword space

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

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

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

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

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

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

8.4 Post-punctuation space

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

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

```
\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.

8.5 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 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.

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

Note that in an actual situation, the `Uppercase` option of the `Letters` feature would probably supply this for you (see [Section 12.2 on page 31](#)).

This package (or the `xltextra` package for \X_{TeX} users) redefines \LaTeX 's `\-` macro such that it adjusts along with the above changes.

8.6 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.

OpenType fonts with optical scaling will exist in several discrete sizes, and these will be selected by X_YTeX *automatically* determined by the current font size. The `OpticalSize` option may be used to specify a different optical size.

For the OpenType font Warnock Pro, we have three optically sized variants: caption, subhead, and display. With `OpticalSize` set to zero, no optical size font substitution is performed:

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

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

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

The `SizeFeatures` feature ([Section 7.5 on page 13](#)) can be used to specify exactly which optical sizes will be used for ranges of font size. For example, something like

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

| Feature | Option | Tag |
|-------------|--------------------|--------------------------|
| Ligatures = | Required | * rlig |
| | NoRequired | rlig <i>(deactivate)</i> |
| | Common | * liga |
| | NoCommon | liga <i>(deactivate)</i> |
| | Contextual | * clig |
| | NoContextual | clig <i>(deactivate)</i> |
| | Rare/Discretionary | dlig |
| | Historical | hlig |
| | TeX | tlig/trep |

* This feature is activated by default.

Table 2: Options for the OpenType font feature ‘Ligatures’.

Part II

OpenType

9 Introduction

TODO: explain OpenType font features.

Some examples of font features have already be seen in previous sections.

Multiple options may be given to any feature that accepts non-numerical input, although doing so will not always work. Some options will override others in generally obvious ways; `Numbers={OldStyle,Lining}` doesn’t make much sense because the two options are mutually exclusive, and \XeTeX will simply use the last option that is specified (in this case using `Lining` over `OldStyle`).

If a feature or an option is requested that the font does not have, a warning is given in the console output. As mentioned in [Section 3.3 on page 4](#) these warnings can be suppressed by selecting the `[quiet]` package option.

10 Complete listing of OpenType font features

10.1 Ligatures

Ligatures refer to the replacement of two separate characters with a specially drawn glyph for functional or aesthetic reasons. The list of options, of which multiple may be selected at one time, is shown in [Table 2](#).

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

| Feature | Option | Tag |
|-----------|---------------------|------|
| Letters = | Uppercase | case |
| | SmallCaps | smcp |
| | PetiteCaps | pcap |
| | UppercaseSmallCaps | c2sc |
| | UppercasePetiteCaps | c2pc |
| | Unicase | unic |

Table 3: Options for the OpenType font feature ‘Letters’.

10.2 Letters

The Letters feature specifies how the letters in the current font will look. OpenType fonts have options: Uppercase, SmallCaps, PetiteCaps, UppercaseSmallCaps, UppercasePetiteCaps, and Unicase.

Petite caps are smaller than small caps. SmallCaps and PetiteCaps turn lowercase letters into the smaller caps letters, whereas the Uppercase . . . options turn the *capital* letters into the smaller caps (good, *e.g.*, for applying to already uppercase acronyms like ‘NASA’). ‘Unicase’ is a weird hybrid of upper and lower case letters.

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

Note that The Uppercase option will (probably) not actually map letters to uppercase.³ It is designed select various uppercase forms for glyphs such as accents and dashes.

```
\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 10.11 on page 24](#)).

10.3 Numbers

The Numbers feature defines how numbers will look in the selected font, accepting options shown in [Table 4](#).

The synonyms Uppercase and Lowercase are equivalent to Lining and OldStyle, respectively. The differences have been shown previously in [Section 7.2 on page 12](#). The Monospaced option is useful for tabular material when digits need to be vertically aligned.

The SlashedZero option replaces the default zero with a slashed version to prevent confusion with an uppercase ‘O’.

³If you want automatic uppercase letters, look to L^AT_EX’s \MakeUppercase command.

| Feature | Option | Tag |
|-----------|--------------------|------|
| Numbers = | Uppercase/Lining | lnum |
| | Lowercase/OldStyle | onum |
| | Proportional | pnum |
| | Monospaced | tnum |
| | SlashedZero | zero |
| | Remap | anum |

Table 4: Options for the OpenType font feature ‘Numbers’.

| Feature | Option | Tag |
|---------------|-------------|----------------------------|
| Contextuals = | Swash | cswh |
| | Alternate | * calt |
| | NoAlternate | calt (<i>deactivate</i>) |
| | WordInitial | init |
| | WordFinal | fina |
| | LineFinal | falt |
| | Inner | medi |

* This feature is activated by default.

Table 5: Options for the OpenType font feature ‘Contextuals’.

The Remap option maps numerals to their Arabic or Farsi equivalents based on the current Language setting (see [Section 10.15 on page 26](#)). This is based on a LuaTeX feature of the luaotfload package, not an OpenType feature.

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

10.4 Contextuals

This feature refers to glyph substitution that vary by their position; things like contextual swashes are accessed here.

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

Historic forms 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.

10.5 Vertical Position

The VerticalPosition feature is used to access things like subscript (Inferior) and superscript (Superior) numbers and letters (and a small amount of punctu-

| Feature | Option | Tag |
|--------------------|--------------------|-----|
| VerticalPosition = | Superior | sup |
| | Inferior | sub |
| | Numerator | num |
| | Denominator | dn |
| | ScientificInferior | sin |
| | Ordinal | ord |

Table 6: Options for the OpenType font feature ‘VerticalPosition’.

ation, sometimes). The Ordinal option will only raise characters that are used in some languages directly after a number.

The ScientificInferior feature will move glyphs further below the baseline than the Inferior feature.

Numerator and Denominator should only be used for creating arbitrary fractions (see next section).

```

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

\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 realscripts package (yet to be released) (or xltextra for X_{La}TeX) redefines the \textsubscript and \textsuperscript commands to use the above font features.

10.6 Fractions

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

```

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

```

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

| Feature | Option | Tag |
|-----------|-----------|------|
| Fractions | = On | frac |
| | Alternate | afrc |

Table 7: Options for the OpenType font feature ‘Fractions’.

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

10.7 StylisticSet

This feature selects a ‘Stylistic Set’ variation, specified numerically. These correspond to OpenType features ss01, ss02, etc.

Here are two examples from the Junicode font⁴; thanks to Adam Buchbinder for the suggestion. This example shows insular (ancient) letterforms:

```
\fontspec{Junicode}
Insular forms. \\
\addfontfeature{StylisticSet=2}
Insular forms. \\
```

And here is shown enlarged minuscules: (capital letters remain unchanged)

```
\fontspec{Junicode}
ENLARGED Minuscules. \\
\addfontfeature{StylisticSet=6}
ENLARGED Minuscules. \\
```

(This is a synonym of the Variant feature for AAT fonts.) See [Section 14 on page 37](#) for a way to assign names to stylistic sets, which should be done on a per-font basis.

10.8 Alternates

Selection of the Alternate feature again must be done numerically.

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

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

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

See [Section 14 on page 37](#) for a way to assign names to alternates, which should be done on a per-font basis.

| Feature Option | Tag |
|-------------------|------|
| Style = Alternate | salt |
| Italic | ital |
| Ruby | ruby |
| Swash | swsh |
| Historic | hist |
| TitlingCaps | titl |
| HorizontalKana | hkna |
| VerticalKana | vkna |

Table 8: Options for the OpenType font feature ‘Style’.

10.9 Style

‘Ruby’ refers to a small optical size, used in Japanese typography for annotations. For fonts with multiple salt OpenType features, use the fontspec Alternate feature instead.

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

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

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

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

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

```
\fontspec{Hiragino Mincho Pro W3}
Latin          \
\addfontfeature{Style={Italic, Ruby}}
Latin
```

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

```
\fontspec{Hiragino Mincho Pro}
\
{\addfontfeature{Style=HorizontalKana}
} \
{\addfontfeature{Style=VerticalKana}
}
```

⁴<http://junicode.sf.net>

| Feature | Option | Tag |
|--------------|--------------|--------------------------|
| Diacritics = | MarkToBase | * mark |
| | NoMarkToBase | mark <i>(deactivate)</i> |
| | MarkToMark | * mkmk |
| | NoMarkToMark | mkmk <i>(deactivate)</i> |
| | AboveBase | * abvm |
| | NoAboveBase | abvm <i>(deactivate)</i> |
| | BelowBase | * blwm |
| | NoBelowBase | blwm <i>(deactivate)</i> |

* This feature is activated by default.

Table 9: Options for the OpenType font feature ‘Diacritics’.

| Feature | Option | Tag |
|-----------|-----------|--------------------------|
| Kerning = | Uppercase | csp |
| | On | * kern |
| | Off | kern <i>(deactivate)</i> |

* This feature is activated by default.

Table 10: Options for the OpenType font feature ‘Kerning’.

10.10 Diacritics

Specifies how diacritics should be placed. These will usually be controlled automatically according to the Script setting.

10.11 Kerning

Specifies how inter-glyph spacing should behave.

As briefly mentioned previously at the end of [Section 12.2 on page 31](#), the Uppercase option will add a small amount of tracking between uppercase letters:

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

10.12 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.

| Feature | Option | Tag |
|------------|-------------|------|
| CJKShape = | Traditional | trad |
| | Simplified | smp1 |
| | JIS1978 | jp78 |
| | JIS1983 | jp83 |
| | JIS1990 | jp90 |
| | Expert | expt |
| | NLC | nlck |

Table 11: Options for the OpenType font feature ‘CJKShape’.

| Feature | Option | Tag |
|------------------|-----------------------|------|
| CharacterWidth = | Proportional | pwid |
| | Full | fwid |
| | Half | hwid |
| | Third | twid |
| | Quarter | qwid |
| | AlternateProportional | palt |
| | AlternateHalf | halt |

Table 12: Options for the OpenType font feature ‘CharacterWidth’.

```

\fontspec{Hiragino Mincho Pro}
{\addfontfeature{CJKShape=Traditional}
}
\addfontfeature{CJKShape=NLC}
}
\addfontfeature{CJKShape=Expert}
}

```

10.13 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.

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]{}% %           \makebox[2.5cm][l]{}% %           \makebox[2.5cm][l]{abcdef}}
\fontspec{Hiragino Mincho Pro}
{\addfontfeature{CharacterWidth=Proportional}\test}\\
{\addfontfeature{CharacterWidth=Full}\test}\\
{\addfontfeature{CharacterWidth=Half}\test}

```

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

```

\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.

10.14 Vertical typesetting

TODO!

10.15 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 13 on the following page](#) and [Table 14 on page 28](#), 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 11.4 on page 30](#).

10.15.1 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 14 on page 37](#)) is the error-checking that is performed when the script or language is requested.

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

Table 13: Defined Scripts for OpenType fonts. Aliased names are shown in adjacent positions marked with red pilcrows (¶), defined in `fontspec.cfg`.

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

Table 14: Defined Languages for OpenType fonts. Note that they are sorted alphabetically *not* by name but by OpenType tag, which is a little irritating, really.

Part III

Fonts and features with X_YTeX

11 X_YTeX-only font features

The features described here are available for any font selected by `fontspec`.

11.1 Mapping

Mapping enables a X_YTeX text-mapping scheme.

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

Using the `tex-text` mapping is also equivalent to writing `Ligatures=TeX`. The use of the latter syntax is recommended for better compatibility with Lua_{TeX} documents.

11.2 Letter spacing

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

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

```
\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 12.2 on page 31](#)).

11.3 Font transformations

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

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

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

```

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

```

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

11.4 Different font technologies: AAT and ICU

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

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

Among some other font features only available through a specific renderer, ICU provides for the `Script` and `Language` features, which allow different font behaviour for different alphabets and languages; see [Section 10.15 on page 26](#) 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.*

11.5 Optical font sizes

Multiple Master fonts are parameterised over orthogonal font axes, allowing continuous selection along such features as weight, width, and optical size (see [Section 13 on page 36](#) for further details). Whereas an OpenType font will have only a few separate optical sizes, a Multiple Master font's optical size can be specified over a continuous range. Unfortunately, this flexibility makes it harder to create an automatic interface through L^AT_EX, and the optical size for a Multiple Master font must always be specified explicitly.

```

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

```

12 Mac OS X's AAT fonts

Mac OS X's font technology began life before the ubiquitous-OpenType era and revolved around the Apple-invented 'AAT' font format. This format had some advantages (and other disadvantages) but it never became widely popular in the font world.

Nonetheless, this is the font format that was first supported by X_YTeX (due to its pedigree on Mac OS X in the first place) and was the first font format supported by fontspec. A number of fonts distributed with Mac OS X are still in the AAT format, such as ‘Skia’. Documents that use these fonts should be compiled with X_YLaTeX using the xdv2pdf driver, as opposed to the default xdvipdfmx. E.g.,

```
xelatex -output-driver="xdv2pdf" filename.tex
```

Mac OS X also supports Multiple Master fonts, which are discussed in [Section 13](#).

12.1 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.

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

12.2 Letters

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.

12.3 Numbers

The Numbers feature defines how numbers will look in the selected font. For both AAT, 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 7.2 on page 12](#).

12.4 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.

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

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

12.5 Vertical position

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

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

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

12.6 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.

In AAT fonts, the ‘fraction slash’ or solidus character, is to be used to create fractions. When `Fractions` are turned On, then only pre-drawn fractions will be used.

```
\fontspec[Fractions=On]{Skia}
12 \quad 56 \ \ % fraction slash
1/2 \quad 5/6 \ % regular slash
```

Using the `Diagonal` option (AAT only), the font will attempt to create the fraction from superscript and subscript characters. This is shown in the following example:

```
\fontspec[Fractions=Diagonal]{Skia}
1357924680 \ \ % fraction slash
\quad 13579/24680 \ % regular slash
```

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

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

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

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


12.7 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.

```
\whiledo{\value{var}9}{% % \stepcounter{trans}% % \fontspec[Variant=\thevar,
Color=005599\thetrans\thetrans]{Zapfino}% % \makebox[0.75\width]{d}% % \stepcounter{var}}
```

See [Section 14 on page 37](#) for a way to assign names to variants, which should be done on a per-font basis.

12.8 Alternates

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

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

See [Section 14 on page 37](#) for a way to assign names to alternates, which should be done on a per-font basis.

12.9 Style

The options of the Style feature are defined in AAT as one of the following: Display, Engraved, IlluminatedCaps, Italic, Ruby,⁵ TallCaps, or TitlingCaps.

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

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

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

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

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

⁵'Ruby' refers to a small optical size, used in Japanese typography for annotations.

```

\fontspec{Hiragino Mincho Pro W3}
Latin      \
\addfontfeature{Style={Italic, Ruby}}
Latin

```

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

```

\fontspec{Hiragino Mincho Pro}
\
{\addfontfeature{Style=HorizontalKana}
} \
{\addfontfeature{Style=VerticalKana}
}

```

12.10 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.

12.11 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.⁶ 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.

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]{}% % \makebox[2.5cm][l]{}% % \makebox[2.5cm][l]{abcdef}}
\fontspec{Hiragino Mincho Pro}
{\addfontfeature{CharacterWidth=Proportional}\test}\
{\addfontfeature{CharacterWidth=Full}\test}\
{\addfontfeature{CharacterWidth=Half}\test}

```

⁶Apple 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.

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

```
\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.

12.12 Annotation

Various Asian fonts are equipped with a more extensive range of numbers and numerals in different forms. For OpenType fonts, the only option supported is `On` and `Off`:

```
\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.

12.13 Vertical typesetting

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

```
\fontspec{Hiragino Mincho Pro}
```

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

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

12.14 Diacritics

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

Some fonts include `O/` *etc.* as diacritics for writing `Ø`. You'll want to turn this feature off (imagine typing `hello/goodbye` and getting 'helløgoodbye' instead!) by

decomposing the two characters in the diacritic into the ones you actually want. I would recommend using the proper T_EX input conventions for obtaining such characters instead.

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

12.15 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.

```
\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
```

13 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:

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

Variations along a multiple master font's optical size axis has been shown previously in [Section 11.5 on page 30](#).

Part IV

Programming interface

This is the beginning of some work to provide some hooks that use fontspec for various macro programming purposes.

14 Defining new features

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

`\newAATfeature`

New AAT features may be created with this command:

```
\newAATfeature{<feature>}{<option>}{<feature code>}{<selector code>}
```

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

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

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

`\newICUfeature`

New OpenType features may be created with this command:

`\newopentypefeature`

```
\newICUfeature{<feature>}{<option>}{<feature tag>}
```

The synonym `\newopentypefeature` is provided for LuaTeX users. In the following example, the Moldavian language (see [Section 10.15 on page 26](#)) must be activated to achieve the effect shown.

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

`\newfontfeature`

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

```
\newfontfeature{<name>}{<input string>}
```

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

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

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

15 Going behind fontspec's back

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

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

Multiple features can either be included in a single declaration:

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

or with multiple declarations:

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

16 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:

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

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

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

```
\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 Color specification. For this type of thing, the `\newfontfeature` command should be used to declare a new, *e.g.*, PurpleColor feature:

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

17 Programming details

In some cases, it is useful to know what the L^AT_EX font family of a specific fontspec font is. After a `\fontspec`-like command, this is stored inside the `\zf@family`

macro. Otherwise, L^AT_EX's own `\f@family` macro can be useful here, too. The raw T_EX font that is defined is stored temporarily in `\zf@basefont`.

The following commands in `expl3` syntax may be used for writing codes that interface with `fontspec`-loaded fonts. All of the following conditionals also exist with T and F suffices as well as TF.

| | |
|--|---|
| <code>\fontspec_if_fontspec_font:TF</code> | Test whether the currently selected font has been loaded by <code>fontspec</code> . |
| <code>\fontspec_if_aat_feature:nnTF</code> | Test whether the currently selected font contains the AAT feature (#1,#2). |
| <code>\fontspec_if_opentype:TF</code> | Test whether the currently selected font is an OpenType font. Always true for LuaTeX fonts. |
| <code>\fontspec_if_feature:nTF</code> | Test whether the currently selected font contains the raw OpenType feature #1. E.g.: <code>\fontspec_if_feature:nTF {pnum} {True} {False}</code> . Returns false if the font is not loaded by <code>fontspec</code> or is not an OpenType font. |
| <code>\fontspec_if_feature:nnnTF</code> | Test whether the currently selected font with raw OpenType script tag #1 and raw OpenType language tag #2 contains the raw OpenType feature tag #3. E.g.: <code>\fontspec_if_feature:nTF {latn} {ROM} {pnum} {True} {False}</code> . Returns false if the font is not loaded by <code>fontspec</code> or is not an OpenType font. |
| <code>\fontspec_if_script:nTF</code> | Returns whether the currently selected font contains the raw OpenType script #1. E.g.: <code>\fontspec_if_script:nTF {latn} {True} {False}</code> . Returns false if the font is not loaded by <code>fontspec</code> or is not an OpenType font. |
| <code>\fontspec_if_language:nTF</code> | Test whether the currently selected font contains the raw OpenType feature #1. E.g.: <code>\fontspec_if_feature:nTF {pnum} {True} {False}</code> . Returns false if the font is not loaded by <code>fontspec</code> or is not an OpenType font. |
| <code>\fontspec_if_language:nnTF</code> | Test whether the currently selected font contains the raw OpenType feature #2 in script #1. E.g.: <code>\fontspec_if_feature:nTF {latn} {pnum} {True} {False}</code> . Returns false if the font is not loaded by <code>fontspec</code> or is not an OpenType font. |
| <code>\fontspec_if_current_script:nTF</code> | Test whether the currently loaded font is using the specified raw OpenType script tag #1. |
| <code>\fontspec_if_current_language:nTF</code> | Test whether the currently loaded font is using the specified raw OpenType language tag #1. |

Part V

The patching/improvement of $\text{\LaTeX} 2_{\epsilon}$ and other packages

Derived originally from `xltxtra`, this package contains patches to various \LaTeX components and third-party packages to improve the default behaviour.

18 Inner emphasis

`fixltx2e`'s method for checking for “inner” emphasis is a little fragile in \XeTeX , because font slant information might be missing from the font. Therefore, we use \LaTeX 's NFSS information, which is more likely to be correct.

19 Unicode footnote symbols

By default \LaTeX defines symbolic footnote characters in terms of commands that don't resolve well; better results can be achieved by using specific unicode characters or proper LICRs with the `xunicode` package.

This problem has been solved by loading the `fixltx2e` and `xunicode` packages in `xltxtra`.

20 Verbatim

Many verbatim mechanisms assume the existence of a ‘visible space’ character that exists in the ASCII space slot of the typewriter font. This character is known in unicode as U+2434: BOX OPEN, which looks like this: ‘`␣`’.

When a unicode typewriter font is used, \LaTeX no longer prints visible spaces for the `verbatim*` environment and `\verb*` command. `xltxtra` fixes this problem by using the correct unicode glyph, and patches the following packages to do the same: `listings`, `fancyvrb`, `moreverb`, and `verbatim`.

In the case that the typewriter font does not contain ‘`␣`’, the Latin Modern Mono font is used as a fallback.

21 Discretionary hyphenation: `\-`

\LaTeX defines the macro `\-` to insert discretionary hyphenation points. However, it is hard-coded in \LaTeX to use the hyphen `-` character. Since `fontspec` makes it easy to change the hyphenation character on a per font basis, it would be nice if `\-` adjusted automatically and now it does.

Part VI

fontspec.sty

22 Implementation

Herein lie the implementation details of this package. Welcome! It was 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.)

```
1 \RequirePackage{expl3,xparse}
2 \ExplSyntaxOn

3 \msg_new:nnn {fontspec} {not-pdftex}
4 {
5   Requires XeTeX or LuaTeX to function!
6 }
7 \xetex_if_engine:F {
8   \luatex_if_engine:TF {
9     \RequirePackage{luatextra}[2010/05/10]
10    \luatexRequireModule{fontspec}
11  }{
12    \msg_error:nn {fontspec} {not-pdftex}
13  }
14 }

\xetex_or_luatex:nn Use #1 if XeTeX or #2 if LuaTeX.
15 \xetex_if_engine:TF
16 { \cs_new_eq:NN \xetex_or_luatex:nn \use_i:nn }
17 { \luatex_if_engine:T
18   { \cs_new_eq:NN \xetex_or_luatex:nn \use_ii:nn }
19 }

\xetex_or_luatex:nnn Use #1 and ({#2} if XeTeX) or ({#3} if LuaTeX).
20 \xetex_if_engine:TF
21 { \cs_new:Npn \xetex_or_luatex:nnn #1#2#3 {#1{#2}} }
22 {
23   \luatex_if_engine:T
24   { \cs_new:Npn \xetex_or_luatex:nnn #1#2#3 {#1{#3}} }
25 }
```

22.1 Bits and pieces

Conditionals

```
26 \newif\ifzf@firsttime
27 \newif\ifzf@nobf
28 \newif\ifzf@noit
29 \newif\ifzf@nosc
30 \newif\ifzf@tfm
```

```

31 \newif\ifzf@atsui
32 \newif\ifzf@icu
33 \newif\ifzf@mm
34 \newif\ifzf@graphite

```

For dealing with legacy maths

```

35 \newif\ifzf@math@euler
36 \newif\ifzf@math@lucida
37 \newif\ifzf@package@euler@loaded

```

For package options:

```

38 \newif\ifzf@configfile
39 \newif\ifzf@math

```

Counters

```

40 \newcount\c@zf@newff
41 \newcount\c@zf@index
42 \newcount\c@zf@script
43 \newcount\c@zf@language
44 \int_new:N \l_fontspec_strnum_int

```

Temporary definition until expl3 has been updated to include this:

```

45 \cs_set:Npn \use:x #1 { \edef\@tempa{#1}\@tempa }
46 \cs_new:Npn \fontspec_setkeys:xx #1#2
47 {
48   \use:x { \exp_not:N \setkeys*[zf]{#1}{#2} }
49 }
50 \cs_new:Npn \fontspec_setkeys:xxx #1#2#3
51 {
52   \use:x { \exp_not:N \setkeys*[zf@#1]{#2}{#3} }
53 }

```

22.2 Error/warning messages

Shorthands for messages:

```

54 \cs_new:Npn \fontspec_error:n { \msg_error:nn {fontspec} }
55 \cs_new:Npn \fontspec_error:nx { \msg_error:nnx {fontspec} }
56 \cs_new:Npn \fontspec_warning:n { \msg_warning:nn {fontspec} }
57 \cs_new:Npn \fontspec_warning:nx { \msg_warning:nnx {fontspec} }
58 \cs_new:Npn \fontspec_warning:nxx { \msg_warning:nnxx {fontspec} }
59 \cs_new:Npn \fontspec_info:n { \msg_info:nn {fontspec} }
60 \cs_new:Npn \fontspec_info:nx { \msg_info:nnx {fontspec} }
61 \cs_new:Npn \fontspec_info:nxx { \msg_info:nnxx {fontspec} }
62 \cs_new:Npn \fontspec_trace:n { \msg_trace:nn {fontspec} }

```

Errors:

```

63 \msg_new:nnn {fontspec} {no-size-info}
64 {
65   Size~ information~ must~ be~ supplied.\
66   For~ example,~ SizeFeatures={Size={8-12},...}.
67 }
68 \msg_new:nnnn {fontspec} {rename-feature-not-exist}

```

```

69 {
70   The~ feature~ #1~ doesn't~ appear~ to~ be~ defined.
71 }
72 {
73   It~ looks~ like~ you're~ trying~ to~ rename~ a~ feature~ that~ doesn't~ exist.
74 }
75 \msg_new:nnn {fontspec} {no-glyph}
76 {
77   '\zf@fontname'~ doesn't~ appear~ to~ have~ the~ glyph~ corresponding~ to~ '#1'.
78 }
79 \msg_new:nnnn {fontspec} {unknown-options}
80 {
81   The~ following~ font~ options~ are~ not~ recognised:\\
82   \space\space\space\space #1
83 }
84 {
85   There~ is~ probably~ a~ typo~ in~ the~ font~ feature~ selection.
86 }
87 \msg_new:nnnn {fontspec} {euler-too-late}
88 {
89   The~ euler~ package~ must~ be~ loaded~ BEFORE~ fontspec.
90 }
91 {
92   fontspec~ only~ overwrites~ euler's~ attempt~ to\\
93   define~ the~ maths~ text~ fonts~ if~ fontspec~ is\\
94   loaded~ after~ euler.~ Type~ return~ to~ proceed\\
95   with~ incorrect~ \string\mathit,~ \string\mathbf,~ etc.
96 }
97

```

Warnings:

```

98 \msg_new:nnn {fontspec} {addfontfeatures-ignored}
99 {
100   \string\addfontfeature (s)~ ignored;\\
101   it~ cannot~ be~ used~ with~ a~ font~ that~ wasn't~ selected~ by~ fontspec.
102 }
103 \msg_new:nnn {fontspec} {feature-option-overwrite}
104 {
105   Option~ '#2'~ of~ font~ feature~ '#1'~ overwritten.
106 }
107 \msg_new:nnn {fontspec} {script-not-exist}
108 {
109   Font~ '\zf@fontname'~ does~ not~ contain~ script~ '#1'.
110 }
111 \msg_new:nnn {fontspec} {aat-feature-not-exist}
112 {
113   '\XKV@tfam=\XKV@tkey'~ feature~ not~ supported\\
114   for~ AAT~ font~ '\zf@fontname'.
115 }
116 \msg_new:nnn {fontspec} {aat-feature-not-exist-in-font}
117 {
118   AAT~ feature~ '\XKV@tfam=\XKV@tkey'~ (#1)~ not~ available\\

```

```

119 in~ font~ '\zf@fontname'.
120 }
121 \msg_new:nnn {fontspec} {icu-feature-not-exist}
122 {
123   '\XKV@tfam=\XKV@tkey'~ feature~ not~ supported\\
124   for~ ICU~ font~ '\zf@fontname'
125 }
126 \msg_new:nnn {fontspec} {icu-feature-not-exist-in-font}
127 {
128   OpenType~ feature~ '\XKV@tfam=\XKV@tkey'~ (#1)~ not~ available\\
129   for~ font~ '\zf@fontname', \\
130   with~ script~ '\l_fontspec_script_name_tl',~ and~ language~ '\l_fontspec_lang_name_tl'.
131 }
132 \msg_new:nnn {fontspec} {no-opticals}
133 {
134   '\zf@fontname'~ doesn't~ appear~ to~ have~ an~ Optical~ Size~ axis.
135 }
136 \msg_new:nnn {fontspec} {language-not-exist}
137 {
138   Language~ '#1'~ not~ available\\
139   for~ font~ '\zf@fontname'\\
140   with~ script~ '\l_fontspec_script_name_tl'.
141 }
142 \msg_new:nnn {fontspec} {only-xetex-feature}
143 {
144   Ignored~ XeTeX~ only~ feature:~ '#1'.
145 }
146 \msg_new:nnn {fontspec} {only-luatex-feature}
147 {
148   Ignored~ LuaTeX~ only~ feature:~ '#1'.
149 }
150 \msg_new:nnn {fontspec} {no-mapping}
151 {
152   Input~ mapping~ not~ (yet?)~ supported~ in~ LuaTeX.
153 }
154 \msg_new:nnn {fontspec} {no-mapping-ligtext}
155 {
156   Input~ mapping~ not~ (yet?)~ supported~ in~ LuaTeX.\\
157   Use~ "Ligatures=TeX"~ instead~ of~ "Mapping=tex-text".
158 }
159 \msg_new:nnn {fontspec} {cm-default-obsolete}
160 {
161   The~ "cm-default"~ package~ option~ is~ obsolete.
162 }

Info messages:
163 \msg_new:nnn {fontspec} {defining-font}
164 {
165   Defining~ font~ family~ for~ '#2'~ with~ options~ [\zf@default@options #1].
166 }
167 \msg_new:nnn {fontspec} {no-font-shape}
168 {

```

```

169 Could~ not~ resolve~ font~ #1~ (it~ probably~ doesn't~ exist).
170 }
171 \msg_new:nnn {fontspec} {set-scale}
172 {
173   \zf@fontname\space scale ~=~ \l_fontspec_scale_tl.
174 }
175 \msg_new:nnn {fontspec} {setup-math}
176 {
177   Adjusting~ the~ maths~ setup~ (use~ [no-math]~ to~ avoid~ this).
178 }

```

22.3 Option processing

```

179 \DeclareOption{cm-default}{
180   \fontspec_warning:n {cm-default-obsolete}
181 }
182 \DeclareOption{math}{\@zf@mathtrue}
183 \DeclareOption{no-math}{\@zf@mathfalse}
184 \DeclareOption{config}{\@zf@configfiletrue}
185 \DeclareOption{no-config}{\@zf@configfilefalse}
186 \DeclareOption{quiet}{
187   \msg_redirect_module:nnn { fontspec } { warning } { info }
188   \msg_redirect_module:nnn { fontspec } { info } { none }
189 }
190 \DeclareOption{silent}{
191   \msg_redirect_module:nnn { fontspec } { warning } { none }
192   \msg_redirect_module:nnn { fontspec } { info } { none }
193 }
194 \ExecuteOptions{config,math}
195 \ProcessOptions*

```

22.4 Packages

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

```

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

```

New for Lua_T_EX, we load a new package called ‘fontspec-patches’ designed to incorporate the hidden but useful parts of the old `xltxtra` package.

```

198 \luatex_if_engine:T { \RequirePackage{fontspec-patches} }

```

22.5 Encodings

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

```

199 \xetex_or luatex:nnn {\tl_set:Nn \zf@enc} {EU1} {EU2}
200 \tl_set:Nn \rmdefault {lmr}
201 \tl_set:Nn \sfdefault {lmss}

```

```

202 \tl_set:Nn \ttdefault {lmtt}
203 \RequirePackage[\zf@enc]{fontenc}
204 \tl_set_eq:NN \UTFencname \zf@enc % for xunicode

```

Dealing with a couple of the problems introduced by babel:

```

205 \tl_set_eq:NN \cyrillicencoding \zf@enc
206 \tl_set_eq:NN \latinencoding \zf@enc
207 \g@addto@macro \document {
208   \tl_set_eq:NN \cyrillicencoding \zf@enc
209   \tl_set_eq:NN \latinencoding \zf@enc
210 }

```

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.

22.6 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 22.8 on page 54](#).

22.6.1 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.

```

211 \DeclareDocumentCommand \fontspec { O{} m } {
212   \fontspec_select:nn{#1}{#2}
213   \fontfamily\zf@family\selectfont
214   \ignorespaces
215 }

```

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

```

216 \DeclareDocumentCommand \setmainfont { O{} m } {
217   \fontspec_select:nn{#1}{#2}
218   \tl_set_eq:NN \rmdefault \zf@family
219   \normalfont
220 }
221 \DeclareDocumentCommand \setsansfont { O{} m } {
222   \fontspec_select:nn{#1}{#2}
223   \tl_set_eq:NN \sfdefault \zf@family
224   \normalfont
225 }

```

```

226 \DeclareDocumentCommand \setmonofont { O{} m } {
227   \fontspec_select:nn{#1}{#2}
228   \tl_set_eq:NN \ttdefault \zf@family
229   \normalfont
230 }

\setromanfont This is the old name for \setmainfont, retained for backwards compatibility.
231 \cs_set_eq:NN \setromanfont \setmainfont

\setmathrm These commands are analogous to \setromanfont and others, but for selecting the
\setmathsf font used for \mathrm, etc. They can only be used in the preamble of the document.
\setboldmathrm \setboldmathrm is used for specifying which fonts should be used in \boldmath.
\setmathtt
232 \DeclareDocumentCommand \setmathrm { O{} m } {
233   \fontspec_select:nn{#1}{#2}
234   \let\zf@rmmaths\zf@family
235 }
236 \DeclareDocumentCommand \setboldmathrm { O{} m } {
237   \fontspec_select:nn{#1}{#2}
238   \let\zf@rmboldmaths\zf@family
239 }
240 \DeclareDocumentCommand \setmathsf { O{} m } {
241   \fontspec_select:nn{#1}{#2}
242   \let\zf@sfmaths\zf@family
243 }
244 \DeclareDocumentCommand \setmathtt { O{} m } {
245   \fontspec_select:nn{#1}{#2}
246   \let\zf@ttmaths\zf@family
247 }
248 \@onlypreamble\setmathrm
249 \@onlypreamble\setboldmathrm
250 \@onlypreamble\setmathsf
251 \@onlypreamble\setmathtt

If the commands above are not executed, then \rmdefault (etc.) will be used.
252 \def\zf@rmmaths{\rmdefault}
253 \def\zf@sfmaths{\sfdefault}
254 \def\zf@ttmaths{\ttdefault}

\newfontfamily This macro takes the arguments of \fontspec with a prepended <instance cmd>
\newfontface (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 repet-
itively (e.g., in a section heading) since continuously calling \zf@fontspec is inef-
ficient because it must parse the option arguments every time.
\zf@fontspec defines a font family and saves its name in \zf@family. This fam-
ily is then used in a typical NFSS \fontfamily declaration, saved in the macro
name specified.
255 \DeclareDocumentCommand \newfontfamily { m O{} m } {
256   \fontspec_select:nn{#2}{#3}
257   \use:x {
258     \exp_not:N \DeclareRobustCommand \exp_not:N #1 {
259       \exp_not:N \fontfamily {\zf@family} \exp_not:N \selectfont

```

```

260     }
261   }
262 }

```

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

```

263 \DeclareDocumentCommand \newfontface { m O{} m } {
264   \newfontfamily #1 [ BoldFont={},ItalicFont={},SmallCapsFont={},#2 ] {#3}
265 }

```

22.6.2 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.

```

266 \DeclareDocumentCommand \defaultfontfeatures {m} {\def\zf@default@options{#1,}}
267 \let\zf@default@options\@empty

```

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

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

The default options are *not* applied (which is why `\zf@default@options` is emptied inside the group; this is allowed as `\zf@family` is globally defined in `\fontspec_select:nn`), 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.

```

268 \DeclareDocumentCommand \addfontfeatures {m} {
269   \ifcsname zf@family@fontdef\f@family\endcsname
270     \begingroup
271       \let\zf@default@options\@empty
272       \use:x {
273         \exp_not:N\fontspec_select:nn
274           {\csname zf@family@options\f@family\endcsname,#1}
275           {\csname zf@family@fontname\f@family\endcsname}
276       }
277     \endgroup
278     \fontfamily\zf@family\selectfont
279   \else
280     \fontspec_warning:n {addfontfeatures-ignored}
281   \fi
282   \ignorespaces
283 }
284 \let\addfontfeature\addfontfeatures

```


22.6.3 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.

```
285 \DeclareDocumentCommand \newfontfeature {mm} {
286   \stepcounter{zf@newff}
287   \cs_set:cpx{+zf-#1}{+zf-\the\c@zf@newff}
288   \define@key[zf]{options}{#1}[]{}
289   \zf@update@family{\csname+zf-#1\endcsname}
290   \zf@update@ff{#2}
291 }
292 }
```

`\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.

```
293 \DeclareDocumentCommand \newAATfeature {mmm} {
294   \unless\ifcsname zf@options@#1\endcsname
295     \zf@define@font@feature{#1}
296   \fi
297   \key@ifundefined[zf]{#1}{#2}{}{
298     \fontspec_warning:nxx {feature-option-overwrite}{#1}{#2}
299   }
300   \zf@define@feature@option{#1}{#2}{#3}{#4}{}
301 }
```

`\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.

```
302 \DeclareDocumentCommand \newICUfeature {mmm} {
303   \unless\ifcsname zf@options@#1\endcsname
304     \zf@define@font@feature{#1}
305   \fi
306   \key@ifundefined[zf]{#1}{#2}{}{
307     \fontspec_warning:nxx {feature-option-overwrite}{#1}{#2}
308   }
309   \zf@define@feature@option{#1}{#2}{}{#3}
310 }
311 \cs_set_eq:NN \newopentypefeature \newICUfeature
```

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

```
312 \DeclareDocumentCommand \aliasfontfeature {mm} {\multi@alias@key{#1}{#2}}
313 \DeclareDocumentCommand \aliasfontfeatureoption {mmm} {
314   \keyval@alias@key[zf@feat]{#1}{#2}{#3}
315 }
```

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

```

316 \DeclareDocumentCommand \newfontscript {mm}
317 {
318   \fontspec_new_script:nn {#1} {#2}
319   \fontspec_new_script:nn {#2} {#2}
320 }

321 \cs_new:Npn \fontspec_new_script:nn #1#2
322 {
323   \define@key[zf@feat]{Script}{#1}[] {
324     \fontspec_check_script:nTF {#2} {
325       \zf@update@family{+script=#1}
326       \tl_set:Nn \l_fontspec_script_tl {#2}
327       \c@zf@script=\l_fontspec_strnum_int\relax
328     }{
329       \fontspec_warning:nx {script-not-exist} {#1}
330     }
331   }
332 }

```

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

```

333 \DeclareDocumentCommand \newfontlanguage {mm}
334 {
335   \fontspec_new_lang:nn {#1} {#2}
336   \fontspec_new_lang:nn {#2} {#2}
337 }

338 \cs_new:Npn \fontspec_new_lang:nn #1#2
339 {
340   \define@key[zf@feat]{Lang}{#1}[] {
341     \fontspec_check_lang:nTF {#2} {
342       \zf@update@family{+lang=#1}
343       \tl_set:Nn \l_fontspec_lang_tl {#2}
344       \c@zf@language=\l_fontspec_strnum_int\relax
345     }{
346       \fontspec_warning:nx {language-not-exist} {#1}
347     }
348   }
349 }

```

`\DeclareFontExtensions` dfont would never be uppercase, right?

```

350 \DeclareDocumentCommand \DeclareFontExtensions {m}
351 {
352   \tl_set:Nx \l_fontspec_extensions_clist { \zap@space #1~\@empty }

```

```

353 }
354 \DeclareFontExtensions{.otf,.ttf,.OTF,.TTF,.ttc,.TTC,.dfont}

```

22.7 Programmer's interface

These functions are not used directly by fontspec when defining fonts; they are designed to be used by other packages who wish to do font-related things on top of fontspec itself.

Because I haven't fully explored how these functions will behave in practise, I am not giving them user-level names. As it becomes more clear which of these should be accessible by document writers, I'll open them up a little more.

All functions are defined assuming that the font to be queried is currently selected as a fontspec font. (I.e., via `\fontspec` or from a `\newfontfamily` macro or from `\setmainfont` and so on.)

```

\fontspec_if_fontspec_font:TF Returns whether the currently selected font has been loaded by fontspec.
355 \prg_new_conditional:Nnn \fontspec_if_fontspec_font: {TF,T,F} {
356   \ifcsname zf@family@fontdef\f@family\endcsname
357     \prg_return_true:
358   \else
359     \prg_return_false:
360   \fi
361 }

\fontspec_if_aat_feature:nnTF Conditional to test if the currently selected font contains the AAT feature (#1,#2).
362 \prg_new_conditional:Nnn \fontspec_if_aat_feature:nn {TF,T,F} {
363   \ifcsname zf@family@fontdef\f@family\endcsname
364     \font\zf@basefont="\use:c{zf@family@fontdef\f@family}"~at~\f@size pt
365     \ifzf@atsui
366       \fontspec_make_AAT_feature_string:nn{#1}{#2}
367       \ifx\@tempa\@empty
368         \prg_return_false:
369       \else
370         \prg_return_true:
371       \fi
372     \else
373       \prg_return_false:
374     \fi
375   \else
376     \prg_return_false:
377   \fi
378 }

\fontspec_if_opentype:TF Returns whether the currently selected font is an OpenType font. Always true for
                          LuaTeX fonts.
379 \prg_new_conditional:Nnn \fontspec_if_opentype: {TF,T,F} {
380   \ifcsname zf@family@fontdef\f@family\endcsname
381     \font\zf@basefont="\csname zf@family@fontdef\f@family\endcsname"~at~\f@size pt
382     \fontspec_set_font_type:
383     \ifzf@icu

```

```

384     \prg_return_true:
385   \else
386     \prg_return_false:
387   \fi
388 \else
389   \prg_return_false:
390 \fi
391 }

```

`\fontspec_if_feature:nTF` Returns whether the currently selected font contains the raw OpenType feature #1. E.g.: `\fontspec_if_feature:nTF {pnum} {True} {False}` Returns false if the font is not loaded by fontspec or is not an OpenType font.

```

392 \prg_new_conditional:Nnn \fontspec_if_feature:n {TF,T,F} {
393   \ifcsname zf@family@fontdef\f@family\endcsname
394     \font\zf@basefont="\csname zf@family@fontdef\f@family\endcsname"~at~\f@size pt
395     \fontspec_set_font_type:
396     \ifzf@icu
397       \int_set:Nn \c@zf@script
398         {\use:c {g_fontspec_script_num_}(\zf@family)_tl}}
399       \int_set:Nn \c@zf@language
400         {\use:c {g_fontspec_lang_num_}(\zf@family)_tl}}
401       \tl_set:Nv \l_fontspec_script_tl {g_fontspec_script_}(\zf@family)_tl}
402       \tl_set:Nv \l_fontspec_lang_tl {g_fontspec_lang_}(\zf@family)_tl}
403       \fontspec_check_ot_feat:nTF {#1} \prg_return_true: \prg_return_false:
404     \else
405       \prg_return_false:
406     \fi
407   \else
408     \prg_return_false:
409   \fi
410 }

```

`\fontspec_if_feature:nnnTF` Returns whether the currently selected font with raw OpenType script tag #1 and raw OpenType language tag #2 contains the raw OpenType feature tag #3. E.g.: `\fontspec_if_feature:nTF {latn} {ROM} {pnum} {True} {False}` Returns false if the font is not loaded by fontspec or is not an OpenType font.

```

411 \prg_new_conditional:Nnn \fontspec_if_feature:nnn {TF,T,F} {
412   \ifcsname zf@family@fontdef\f@family\endcsname
413     \font\zf@basefont="\csname zf@family@fontdef\f@family\endcsname"~at~\f@size pt
414     \fontspec_set_font_type:
415     \ifzf@icu
416       \fontspec_iv_str_to_num:n{#1} \c@zf@script = \l_fontspec_strnum_int \relax
417       \fontspec_iv_str_to_num:n{#2} \c@zf@language = \l_fontspec_strnum_int \relax
418       \fontspec_check_ot_feat:nTF {#3} \prg_return_true: \prg_return_false:
419     \else
420       \prg_return_false:
421     \fi
422   \else
423     \prg_return_false:
424   \fi
425 }

```

`\fontspec_if_script:nTF` Returns whether the currently selected font contains the raw OpenType script #1. E.g.: `\fontspec_if_script:nTF {latn} {True} {False}` Returns false if the font is not loaded by fontspec or is not an OpenType font.

```

426 \prg_new_conditional:Nnn \fontspec_if_script:n {TF,T,F} {
427   \ifcsname zf@family@fontdef\f@family\endcsname
428     \font\zf@basefont="\csname zf@family@fontdef\f@family\endcsname"~at~\f@size pt
429     \fontspec_set_font_type:
430     \ifzf@icu
431       \fontspec_check_script:nTF {#1} \prg_return_true: \prg_return_false:
432     \else
433       \prg_return_false:
434     \fi
435   \else
436     \prg_return_false:
437   \fi
438 }

```

`\fontspec_if_language:nTF` Returns whether the currently selected font contains the raw OpenType feature #1. E.g.: `\fontspec_if_feature:nTF {+pnum} {True} {False}` Returns false if the font is not loaded by fontspec or is not an OpenType font.

```

439 \prg_new_conditional:Nnn \fontspec_if_language:n {TF,T,F} {
440   \ifcsname zf@family@fontdef\f@family\endcsname
441     \font\zf@basefont="\csname zf@family@fontdef\f@family\endcsname"~at~\f@size pt
442     \fontspec_set_font_type:
443     \ifzf@icu
444       \tl_set:Nv \l_fontspec_script_tl {g_fontspec_script_(\zf@family)_tl}
445       \int_set:Nn \c@zf@script
446         {\use:c {g_fontspec_script_num_(\zf@family)_tl}}
447       \fontspec_check_lang:nTF {#1} \prg_return_true: \prg_return_false:
448     \else
449       \prg_return_false:
450     \fi
451   \else
452     \prg_return_false:
453   \fi
454 }

```

`\fontspec_if_language:nnTF` Returns whether the currently selected font contains the raw OpenType feature #2 in script #1. E.g.: `\fontspec_if_feature:nTF {+pnum} {True} {False}` Returns false if the font is not loaded by fontspec or is not an OpenType font.

```

455 \prg_new_conditional:Nnn \fontspec_if_language:nn {TF,T,F} {
456   \ifcsname zf@family@fontdef\f@family\endcsname
457     \font\zf@basefont="\csname zf@family@fontdef\f@family\endcsname"~at~\f@size pt
458     \fontspec_set_font_type:
459     \ifzf@icu
460       \tl_set:Nn \l_fontspec_script_tl {#1}
461       \fontspec_iv_str_to_num:n{#1} \c@zf@script = \l_fontspec_strnum_int \relax
462       \fontspec_check_lang:nTF {#2} \prg_return_true: \prg_return_false:
463     \else
464       \prg_return_false:
465     \fi

```

```

466 \else
467 \prg_return_false:
468 \fi
469 }

```

`\fontspec_if_current_script:nTF` Returns whether the currently loaded font is using the specified raw OpenType script tag #1.

```

470 \prg_new_conditional:Nnn \fontspec_if_current_script:n {TF,T,F} {
471 \ifcsname zf@family@fontdef\f@family\endcsname
472 \font\zf@basefont="\csname zf@family@fontdef\f@family\endcsname"~at~\f@size pt
473 \fontspec_set_font_type:
474 \ifzf@icu
475 \tl_if_eq:nvTF {#1} {g_fontspec_script_(\zf@family)_tl}
476 {\prg_return_true:} {\prg_return_false:}
477 \else
478 \prg_return_false:
479 \fi
480 \else
481 \prg_return_false:
482 \fi
483 }

```

`\fontspec_if_current_language:nTF` Returns whether the currently loaded font is using the specified raw OpenType language tag #1.

```

484 \prg_new_conditional:Nnn \fontspec_if_current_language:n {TF,T,F} {
485 \ifcsname zf@family@fontdef\f@family\endcsname
486 \font\zf@basefont="\csname zf@family@fontdef\f@family\endcsname"~at~\f@size pt
487 \fontspec_set_font_type:
488 \ifzf@icu
489 \tl_if_eq:nvTF {#1} {g_fontspec_lang_(\zf@family)_tl}
490 {\prg_return_true:} {\prg_return_false:}
491 \else
492 \prg_return_false:
493 \fi
494 \else
495 \prg_return_false:
496 \fi
497 }

```

Need this:

```

498 \cs_generate_variant:Nn \tl_if_eq:nnTF {nv}

```

22.8 Internal macros

The macros from here in are used internally by all those defined above. They are not designed to remain consistent between versions.

`\fontspec_select:nn` This is the command that defines font families for use, the underlying procedure of all `\fontspec`-like commands. Given a list of font features (#1) for a requested font (#2, stored in `\zf@fontname` globally for the `\zf@make@aat@feature@string`

macro), it will define an NFSS family for that font and put the family name into `\zf@family`.

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

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

```
499 \cs_set:Npn \fontspec_select:nn #1#2 {
500   \begingroup
501   \fontspec_init:
```

`\zf@fontname` is used as the generic name of the font being defined. `\zf@family@long` is the unique identifier of the font with all its features. `\zf@up` is the font specifically to be used as the upright font.

```
502   \edef\zf@fontname{#2}
503   \let\zf@family@long\zf@fontname
504   \let\zf@up\zf@fontname
```

Detect if external fonts are to be used, possibly automatically, and parse `fontspec` features for bold/italic fonts and their features.

```
505   \fontspec_if_detect_external:nT {#2}
506   { \setkeys[zf]{preparse-external}{ExternalLocation} }
507   \fontspec_setkeys:xx {preparse-external} {\zf@default@options #1}
```

When `\zf@fontname` is augmented with a prefix or whatever to create the name of the upright font (`\zf@up`), this latter is the new 'general font name' to use.

```
508   \let\zf@fontname\zf@up
509   \fontspec_setkeys:xx {preparse} {\XKV@rm}
510   \let\zf@font@feat\XKV@rm

511   \global\font\zf@basefont="\fontspec_fullname:n {\zf@up}"~at~\f@size pt
512   \fontspec_set_font_type:
513   \global\font\zf@basefont="\fontspec_fullname:n {\zf@up}"~at~\f@size pt
```

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

```
514   \zf@firsttimetrue
515   \ifzf@icu
516     \fontspec_setkeys:xxx {feat} {Script} {\l_fontspec_script_name_tl}
517     \fontspec_setkeys:xxx {feat} {Lang}   {\l_fontspec_lang_name_tl}
518   \fi
519   \fontspec_get_features:n{\zf@font@feat}
520   \zf@firsttimefalse
```

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

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

```

521 \unless\ifcsname zf@UID@zf@family@long\endcsname
522 \ifcsname c@zf@famc@#2\endcsname
523 \expandafter\global\expandafter\advance
524 \csname c@zf@famc@#2\endcsname\@ne
525 \else
526 \expandafter\global\expandafter\newcount
527 \csname c@zf@famc@#2\endcsname
528 \fi
529 \edef\@tempa{#2~}
530 \cs_gset:cpx{zf@UID@zf@family@long}{
531 \expandafter\zap@space\@tempa\@empty
532 (\expandafter\the\csname c@zf@famc@#2\endcsname)
533 }
534 \fi
535 \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`.

```

536 \unless\ifcsname zf@family@fontname\zf@family\endcsname
537 \fontspec_info:nxx {defining-font} {#1} {#2}
538 \tl_gset:cx {zf@family@fontname\zf@family} {#2}
539 \tl_gset:cx {zf@family@options\zf@family} {\zf@default@options #1}
540 \tl_gset:cx {zf@family@fontdef\zf@family} {
541 \fontspec_fullname:n {\zf@fontname} : \l_fontspec_pre_feat_tl \l_fontspec_rawfeatures_sclist
542 }
543 \tl_gset:cx {g_fontspec_script_num_(\zf@family)_tl}
544 { \int_use:N \c@zf@script }
545 \tl_gset:cx {g_fontspec_lang_num_(\zf@family)_tl}
546 { \int_use:N \c@zf@language }
547 \tl_gset_eq:cN {g_fontspec_script_(\zf@family)_tl} \l_fontspec_script_tl
548 \tl_gset_eq:cN {g_fontspec_lang_(\zf@family)_tl} \l_fontspec_lang_tl
```

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

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

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

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

```

549 \DeclareFontFamily{\zf@enc}{\zf@family}{}
550 \zf@make@font@shapes{\zf@fontname}
```



```
551      {\mddefault}{\updefault}{\zf@font@feat\zf@up@feat}
```

And the different shapes are set accordingly.

```
552      \fontspec_set_bold:
553      \fontspec_set_italic:
554      \fontspec_set_slanted:
555      \fontspec_set_bold_italic:
556      \fontspec_set_bold_slanted:
557      \fi
558      \endgroup
559 }
```

`\fontspec_if_detect_external:nT` Check if either the fontname ends with a known font extension.

```
560 \prg_new_conditional:Nnn \fontspec_if_detect_external:n {T}
561 {
562   \clist_map_inline:Nn \l_fontspec_extensions_clist
563   {
564     \bool_set_false:N \l_tmpa_bool
565     \tl_if_in:nnT {#1 = end_of_string} {##1 = end_of_string}
566     { \bool_set_true:N \l_tmpa_bool \clist_map_break: }
567   }
568   \bool_if:NTF \l_tmpa_bool \prg_return_true: \prg_return_false:
569 }
```

`\fontspec_fullname:n` Constructs the complete font name based on a common piece of info.

```
570 \cs_set:Npn \fontspec_fullname:n #1 {
571   \fontspec_namewrap:n { #1 \l_fontspec_extension_tl }
572   \l_fontspec_renderer_tl
573   \l_fontspec_optical_size_tl
574 }
```

`\fontspec_set_bold:` Again, the extra bold options defined with BoldFeatures are appended to the generic font features. Then, the bold font is defined either as the ATS default (`\zf@make@font@shapes'` optional argument is to check if there actually is one; if not, the bold NFSS series is left undefined) or with the font specified with the BoldFont feature.

```
575 \cs_new:Npn \fontspec_set_bold: {
576   \unless\ifzf@nobf
577     \ifx\zf@bf\@empty
578       \zf@make@font@shapes[\zf@fontname]{/B}
579       {\bfdefault}{\updefault}{\zf@font@feat\zf@bf@feat}
580     \else
581       \zf@make@font@shapes{\zf@bf}
582       {\bfdefault}{\updefault}{\zf@font@feat\zf@bf@feat}
583     \fi
584   \fi
585 }
```

`\fontspec_set_italic:` And italic in the same way:

```
586 \cs_new:Npn \fontspec_set_italic: {
587   \unless\ifzf@noit
```

```

588 \ifx\zf@it\empty
589 \zf@make@font@shapes[\zf@fontname]{/I}
590 {\mddefault}{\itdefault}{\zf@font@feat\zf@it@feat}
591 \else
592 \zf@make@font@shapes{\zf@it}
593 {\mddefault}{\itdefault}{\zf@font@feat\zf@it@feat}
594 \fi
595 \fi
596 }

```

\fontspec_set_slanted: And slanted but only if requested:

```

597 \cs_new:Npn \fontspec_set_slanted: {
598 \ifx\zf@sl\empty\else
599 \zf@make@font@shapes{\zf@sl}
600 {\mddefault}{\sldefault}{\zf@font@feat\zf@sl@feat}
601 \fi
602 }

```

\fontspec_set_bold_italic: 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.

```

603 \cs_new:Npn \fontspec_set_bold_italic: {
604 \@tempwatrue
605 \ifzf@nobf\@tempwafalse\fi
606 \ifzf@noit\@tempwafalse\fi
607 \if@tempswa
608 \ifx\zf@bfit\empty
609 \ifx\zf@bf\empty
610 \ifx\zf@it\empty
611 \zf@make@font@shapes[\zf@fontname]{/BI}
612 {\bfdefault}{\itdefault}{\zf@font@feat\zf@bfit@feat}
613 \else
614 \zf@make@font@shapes[\zf@it]{/B}
615 {\bfdefault}{\itdefault}{\zf@font@feat\zf@bfit@feat}
616 \fi
617 \else
618 \zf@make@font@shapes[\zf@bf]{/I}
619 {\bfdefault}{\itdefault}{\zf@font@feat\zf@bfit@feat}
620 \fi
621 \else
622 \zf@make@font@shapes{\zf@bfit}
623 {\bfdefault}{\itdefault}{\zf@font@feat\zf@bfit@feat}
624 \fi
625 \fi
626 }

```

\fontspec_set_bold_slanted: And bold slanted, again, only if requested:

```

627 \cs_new:Npn \fontspec_set_bold_slanted: {
628 \ifx\zf@bfs\empty
629 \ifx\zf@sl\empty\else

```

```

630     \zf@make@font@shapes[\zf@sl]{/B}
631     {\bfdefault}{\sldefault}{\zf@font@feat\zf@bfs1@feat}
632   \fi
633   \else
634     \zf@make@font@shapes{\zf@bfs1}
635     {\bfdefault}{\sldefault}{\zf@font@feat\zf@bfs1@feat}
636   \fi
637 }

```

22.8.1 Fonts

`\fontspec_set_font_type:` 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.

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.

```

638 \xetex_or luatex:nnn { \cs_new:Npn \fontspec_set_font_type: }
639 {
640   \zf@tfmfalse \zf@atsuifalse \zf@icufalse \zf@mmfalse \zf@graphitefalse
641   \ifcase\XeTeXfonttype\zf@basefont
642     \zf@tfmtrue
643   \or
644     \zf@atsui true
645     \ifnum\XeTeXcountvariations\zf@basefont \c_zero
646       \zf@mmtrue
647     \fi
648   \or
649     \zf@icutrue
650   \fi

```

If automatic, the `\l_fontspec_renderer_tl` token list 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.

```

651   \tl_if_empty:NT \l_fontspec_renderer_tl {
652     \ifzf@atsui
653       \tl_set:Nn \l_fontspec_renderer_tl {/AAT}
654     \else\ifzf@icu
655       \tl_set:Nn \l_fontspec_renderer_tl {/ICU}
656     \fi\fi
657   }
658 }
659 {
660   \zf@icutrue
661 }

```

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

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

The optional first argument is used when making the font shapes for bold, italic, and bold italic fonts using X_YTeX's auto-recognition with #2 as /B, /I, and /BI font name suffixes. If no such font is found, it falls back to the original font name, in which case this macro doesn't proceed and the font shape is not created for the NFSS.

```

662 \newcommand*{\zf@make@font@shapes}[5][[]{
663   \begingroup
664   \edef\@tempa{#1}
665   \unless\ifx\@tempa\@empty
666     \font\@tempfonta="\fontspec_fullname:n {#1}"~at~\f@size pt
667     \edef\@tempa{\fontname\@tempfonta}
668   \fi
669   \font\@tempfontb="\fontspec_fullname:n {#1#2}"~at~\f@size pt
670   \edef\@tempb{\fontname\@tempfontb}
671   \ifx\@tempa\@tempb
672     \fontspec_info:nx {no-font-shape} {#1#2}
673   \else
674     \edef\zf@fontname{#1#2}
675     \let\zf@basefont\@tempfontb
676     \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 22.10 on page 86](#) for the code that enables this usage.

```

677   \ifx\zf@sc\@empty
678     \unless\ifzf@nosc
679       \zf@make@smallcaps
680       \unless\ifx\zf@smallcaps\@empty
681         \zf@DeclareFontShape[\zf@smallcaps]{#3}
682         {\ifx#4\itdefault\sidefault\else\scdefault\fi}{#5\zf@sc@feat}
683       \fi
684     \fi
685   \else
686     \edef\zf@fontname{\zf@sc}
687     \zf@DeclareFontShape{#3}
688     {\ifx#4\itdefault\sidefault\else\scdefault\fi}{#5\zf@sc@feat}
689   \fi
690 \fi
691 \endgroup
692 }

```

Note that the test for italics to choose the `\sidefault` shape only works while `\fontspec_select:nn` passes single tokens to this macro...

```

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

```

#3 : Font shape

#4 : Font features

Wrapper for \DeclareFontShape.

```
693 \newcommand\zf@DeclareFontShape[4][]{
694   \clist_if_empty:NTF \l_fontspec_sizefeat_clist
695   {
696     \fontspec_get_features:n{#4}
697     \tl_set:Nx \l_fontspec_nfss_tl {
698       - \l_fontspec_scale_tl "
699       \fontspec_fullname:n {\zf@fontname} :
700       \l_fontspec_pre_feat_tl \l_fontspec_rawfeatures_sclist #1 "
701     }
702   }
```

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

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

```
703 {
704   \tl_clear:N \l_fontspec_nfss_tl
705   \clist_map_inline:Nn \l_fontspec_sizefeat_clist {
706     \tl_clear:N \l_fontspec_size_tl
707     \tl_set_eq:NN \l_fontspec_sizedfont_tl \zf@fontname
708     \fontspec_setkeys:xx {sizing} { \expandafter \@firstofone ##1 }
709     \tl_if_empty:NT \l_fontspec_size_tl { \fontspec_error:n {no-size-info} }
710     \fontspec_get_features:n{#4,\XKV@rm}
711     \tl_put_right:Nx \l_fontspec_nfss_tl {
712       \l_fontspec_size_tl \l_fontspec_scale_tl
713       " \fontspec_fullname:n { \l_fontspec_sizedfont_tl }
714       : \l_fontspec_pre_feat_tl \l_fontspec_rawfeatures_sclist #1 "
715     }
716   }
717 }
```

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

```
718 \use:x{
719   \exp_not:N\DeclareFontShape{\zf@enc}{\zf@family}{#2}{#3}
720   {\l_fontspec_nfss_tl}{\zf@adjust}
721 }
```

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

```
722 \tl_if_eq:xxT {#3} {\itdefault}
723 {
724   \use:x {
725     \exp_not:N \DeclareFontShape {\zf@enc}{\zf@family}{#2}{\sldefault}
726     {-ssub*\zf@family/#2/\itdefault}{\zf@adjust}
727   }
728 }
729 }
```

`\l_fontspec_pre_feat_tl` These are the features always applied to a font selection before other features.

```

730 \xetex_or luatex:nnn { \tl_set:Nn \l_fontspec_pre_feat_tl }
731 {
732   \ifzf@icu
733     script = \l_fontspec_script_tl ;
734     language = \l_fontspec_lang_tl ;
735   \fi
736 }
737 {
738   mode = \l_fontspec_mode_tl ;
739   script = \l_fontspec_script_tl ;
740   language = \l_fontspec_lang_tl ;
741 }

```

`\zf@update@family` This macro is used to build up a complex family name based on its features.
`\zf@firsttime` is set true in `\fontspec_select:nn` only the first time `\f@get@feature@requests` is called, so that the family name is only created once.

```

742 \newcommand*{\zf@update@family}[1]{
743   \ifzf@firsttime
744     \xdef\zf@family@long{\zf@family@long#1}
745   \fi
746 }

```

22.8.2 Features

`\fontspec_get_features:n` 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.

```

747 \cs_set:Npn \fontspec_get_features:n #1 {
748   \let\l_fontspec_rawfeatures_sclist \@empty
749   \tl_clear:N \l_fontspec_scale_tl
750   \let\zf@adjust \@empty
751   \fontspec_setkeys:xx {options} {#1}
752   \tl_if_empty:NF \XKV@rm {
753     \fontspec_error:nx {unknown-options} { \exp_not:V \XKV@rm }
754   }
755   \PackageInfo{fontspec}{Raw~ font~ features~ "\l_fontspec_rawfeatures_sclist"}
756 }

```

`\fontspec_init:` Initialisations that either need to occur globally: (all setting of these variables is done locally inside a group)

```

757 \tl_clear:N \zf@bf
758 \tl_clear:N \zf@it
759 \tl_clear:N \zf@bfit
760 \tl_clear:N \zf@sl
761 \tl_clear:N \zf@bfsl
762 \tl_clear:N \zf@sc
763 \tl_clear:N \zf@up@feat
764 \tl_clear:N \zf@bf@feat
765 \tl_clear:N \zf@it@feat

```

```

766 \tl_clear:N \zf@bfit@feat
767 \tl_clear:N \zf@sl@feat
768 \tl_clear:N \zf@bfs1@feat
769 \tl_clear:N \zf@sc@feat
770 \clist_clear:N \l_fontspec_sizefeat_clist

```

Or once per fontspec font invocation: (Some of these may be redundant. Check whether they're assigned to globally or not.)

```

771 \newcommand*\fontspec_init:{
772   \zf@icufalse
773   \cs_set_eq:NN \fontspec_namewrap:n \use:n
774   \tl_clear:N \l_fontspec_optical_size_tl
775   \tl_clear:N \l_fontspec_renderer_tl
776   \tl_set:Nn \l_fontspec_script_name_tl {Latin}
777   \tl_set:Nn \l_fontspec_script_tl {latn}
778   \tl_set:Nn \l_fontspec_lang_name_tl {Default}
779   \tl_set:Nn \l_fontspec_lang_tl {DFLT}
780   \luatex_if_engine:T {
781     \tl_set:Nn \l_fontspec_mode_tl {node}
782     \luatexprehyphenchar = '\- % fixme
783     \luatexposthyphenchar = 0 % fixme
784     \luatexpreexhyphenchar = 0 % fixme
785     \luatexpostexhyphenchar = 0 % fixme
786   }
787 }

```

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

```

788 \newcommand*\zf@make@smallcaps{
789   \let\zf@smallcaps\@empty
790   \xetex_or luatex:nn
791   {
792     \ifzf@atsui
793       \fontspec_make_AAT_feature_string:nn{3}{3}
794       \unless\ifx\@tempa\@empty
795         \edef\zf@smallcaps{\@tempa;}
796       \fi
797     \fi
798     \ifzf@icu
799       \fontspec_check_ot_feat:nT {+smcp} {\edef\zf@smallcaps{+smcp;}}
800     \fi
801   }
802   {
803     \fontspec_check_ot_feat:nT {+smcp} {\edef\zf@smallcaps{+smcp;}}
804   }
805 }

```

`\sclist_put_right:Nn` I'm hardly going to write an 'sclist' module but a couple of functions are necessary.

```

806 \cs_new:Npn \sclist_put_right:Nn #1#2 {
807   \tl_if_empty:NT #1 {
808
809   }

```

810 }

\zf@update@ff \l_fontspec_rawfeatures_sclist 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. Font features are separated by semicolons.

```
811 \newcommand*\zf@update@ff[1]{
812   \unless\ifzf@firsttime
813     \xdef\l_fontspec_rawfeatures_sclist{\l_fontspec_rawfeatures_sclist #1;}
814   \fi
815 }
```

\fontspec_make_feature:nnn This macro is called by each feature key selected, and runs according to which type of font is selected.

```
816 \cs_new:Npn \fontspec_make_feature:nnn #1#2#3 {
817   \xetex_or luatex:nn
818   {
819     \ifzf@atsui
820       \fontspec_make_AAT_feature:nn {#1}{#2}
821     \fi
822     \ifzf@icu
823       \fontspec_make_ICU_feature:n {#3}
824     \fi
825   }
826   {
827     \fontspec_make_ICU_feature:n {#3}
828   }
829 }
830 \cs_new:Npn \fontspec_make_AAT_feature:nn #1#2 {
831   \tl_if_empty:nTF {#1}
832   {
833     \fontspec_warning:n {aat-feature-not-exist}
834   }
835   {
836     \fontspec_make_AAT_feature_string:nn {#1}{#2}
837     \ifx\@tempa\@empty
838       \fontspec_warning:nx {aat-feature-not-exist-in-font} {#1,#2}
839     \else
840       \zf@update@family{+#1,#2}
841       \zf@update@ff\@tempa
842     \fi
843   }
844 }
845 \cs_new:Npn \fontspec_make_ICU_feature:n #1 {
846   \tl_if_empty:nTF {#1}
847   {
848     \fontspec_warning:n {icu-feature-not-exist}
849   }
850   {
851     \fontspec_check_ot_feat:nTF {#1} {
852       \zf@update@family{#1}
853       \zf@update@ff{#1}

```



```

854     }{
855     \fontspec_warning:nx {icu-feature-not-exist-in-font} {#1}
856     }
857   }
858 }

```

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

```

\zf@define@feature@option 859 \newcommand*\zf@define@font@feature[1]{
860   \define@key[zf]{options}{#1}{\setkeys[zf@feat]{#1}{##1}}
861 }
862 \newcommand*\zf@define@feature@option[5]{
863   \define@key[zf@feat]{#1}{#2}[]{\fontspec_make_feature:nnn{#3}{#4}{#5}}
864 }

```

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

```

865 \newcommand*\keyval@alias@key[4][KV]{
866   \cs_set_eq:cc{#1@#2@#4}{#1@#2@#3}
867   \cs_set_eq:cc{#1@#2@#4@default}{#1@#2@#3@default}
868 }

```

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

```

869 \newcommand*\multi@alias@key[2]{
870   \key@ifundefined[zf]{options}{#1}
871   {
872     \key@ifundefined[zf]{preparse}{#1}
873     {
874       \key@ifundefined[zf]{preparse-external}{#1}
875       { \fontspec_warning:nx {rename-feature-not-exist} {#1} }
876       { \keyval@alias@key[zf]{preparse-external}{#1}{#2} }
877     }
878     { \keyval@alias@key[zf]{preparse}{#1}{#2} }
879   }
880   { \keyval@alias@key[zf]{options}{#1}{#2} }
881 }

```

`pec_make_AAT_feature_string:nn` 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 63).

```

882 \cs_new:Npn \fontspec_make_AAT_feature_string:nn #1#2 {
883   \edef\@tempa{\XeTeXfeaturename\zf@basefont #1}
884   \unless\ifx\@tempa\@empty

```

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

```

885   \ifnum\XeTeXisexclusivefeature\zf@basefont #10
886     \edef\@tempb{\XeTeXselectorname\zf@basefont #1\space #2}

```

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

```

887   \else

```

```

888     \unless\ifodd #2
889     \edef\@tempb{\XeTeXselectorname\zf@basefont #1\space #2}

```

If the selector is *odd*, it corresponds to switching the feature off. But X_YTeX 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.

```

890     \else
891     \edef\@tempb{
892     \XeTeXselectorname\zf@basefont #1\space \numexpr#2-1\relax
893     }
894     \unless\ifx\@tempb\@empty
895     \edef\@tempb{!\@tempb}
896     \fi
897     \fi
898     \fi

```

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

```

899     \unless\ifx\@tempb\@empty
900     \edef\@tempa{\@tempa=\@tempb}
901     \else
902     \let\@tempa\@empty
903     \fi
904     \fi
905 }

```

`\fontspec_iv_str_to_num:n` This macro takes a four character string and converts it to the numerical representation required for X_YTeX OpenType script/language/feature purposes. The output is stored in `\l_fontspec_strnum_int`.

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 \@emptys and anything beyond four chars is snipped. The \@emptys then are used to reconstruct the spaces in the string to number calculation.

The variant `\fontspec_v_str_to_num:n` 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 `\fontspec_iv_str_to_num:n`.

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.

```

906 \cs_set:Npn \fontspec_iv_str_to_num:n #1 {
907   \fontspec_iv_str_to_num:w #1 \@empty \@empty \q_nil
908 }
909 \cs_set:Npn \fontspec_iv_str_to_num:w #1#2#3#4#5 \q_nil {
910   \int_set:Nn \l_fontspec_strnum_int {
911     '#1 * "1000000
912     + '#2 * "10000
913     + \ifx \@empty #3 32 \else '#3 \fi * "100
914     + \ifx \@empty #4 32 \else '#4 \fi

```

```

915 }
916 }
917 \cs_set:Npn \fontspec_v_str_to_num:n #1 {
918   \bool_if:nTF
919     {
920       \tl_if_head_eq_charcode_p:nN {#1} {+} ||
921       \tl_if_head_eq_charcode_p:nN {#1} {-}
922     }
923     {
924       \exp_after:wN \fontspec_iv_str_to_num:n
925       \exp_after:wN { \use_none:n #1 }
926     }
927     { \fontspec_iv_str_to_num:n {#1} }
928 }

```

`\fontspec_check_script:nTF` This macro takes an OpenType script tag and checks if it exists in the current font. The output boolean is `\@tempswattrue`. `\l_fontspec_strnum_int` is used to store the number corresponding to the script tag string.

```

929 \xetex_or luatex:nnn {\prg_new_conditional:Nnn \fontspec_check_script:n {TF}}
930 {
931   \fontspec_iv_str_to_num:n{#1}
932   \@tempcntb\XeTeXOTcountscripts\zf@basefont
933   \c@zf@index\z@ \@tempwafalse
934   \loop\ifnum\c@zf@index\@tempcntb
935     \ifnum\XeTeXOTscripttag\zf@basefont\c@zf@index=\l_fontspec_strnum_int
936       \@tempwattrue
937       \c@zf@index\@tempcntb
938     \else
939       \advance\c@zf@index\@ne
940     \fi
941   \repeat
942   \if@tempswa \prg_return_true: \else: \prg_return_false: \fi:
943 }
944 {
945   \directlua{fontspec.check_ot_script("zf@basefont", "#1")}
946   \if@tempswa \prg_return_true: \else: \prg_return_false: \fi:
947 }

```

`\fontspec_check_lang:nTF` This macro takes an OpenType language tag and checks if it exists in the current font/script. The output boolean is `\@tempswattrue`. `\l_fontspec_strnum_int` 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'.

```

948 \xetex_or luatex:nnn {\prg_new_conditional:Nnn \fontspec_check_lang:n {TF}}
949 {
950   \fontspec_iv_str_to_num:n{#1}
951   \@tempcntb\XeTeXOTcountlanguages\zf@basefont\c@zf@script
952   \c@zf@index\z@
953   \@tempwafalse
954   \loop\ifnum\c@zf@index\@tempcntb
955     \ifnum\XeTeXOTlanguagetag\zf@basefont\c@zf@script\c@zf@index=\l_fontspec_strnum_int

```

```

956         \@tempswatrue
957         \c@zf@index\@tempcntb
958     \else
959         \advance\c@zf@index\@ne
960     \fi
961 \repeat
962 \if@tempswa \prg_return_true: \else: \prg_return_false: \fi:
963 }
964 {
965     \directlua{
966         fontspec.check_ot_lang( "zf@basefont", "#1", "\l_fontspec_script_tl" )
967     }
968     \if@tempswa \prg_return_true: \else: \prg_return_false: \fi:
969 }

```

`\fontspec_check_ot_feat:nTF` This macro takes an OpenType feature tag and checks if it exists in the current font/script/language. The output boolean is `\@tempswa`. `\l_fontspec_strnum_int` 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'.

```

970 \xetex_or luatex:nnn
971 { \prg_new_conditional:Nnn \fontspec_check_ot_feat:n {TF,T} }
972 {
973     \@tempcntb\XeTeXOTcountfeatures\zf@basefont\c@zf@script\c@zf@language
974     \fontspec_v_str_to_num:n {#1}
975     \c@zf@index\z@
976     \@tempwafalse
977     \loop\ifnum\c@zf@index\@tempcntb
978         \ifnum\XeTeXOTfeaturetag\zf@basefont\c@zf@script\c@zf@language
979             \c@zf@index=\l_fontspec_strnum_int
980             \@tempswatrue
981             \c@zf@index\@tempcntb
982         \else
983             \advance\c@zf@index\@ne
984         \fi
985     \repeat
986     \if@tempswa \prg_return_true: \else: \prg_return_false: \fi:
987 }
988 {
989     \directlua{
990         fontspec.check_ot_feat(
991             "zf@basefont", "#1",
992             "\l_fontspec_lang_tl", "\l_fontspec_script_tl"
993         )
994     }
995     \if@tempswa \prg_return_true: \else: \prg_return_false: \fi:
996 }

```

22.9 keyval definitions

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

22.9.1 Pre-parsing naming information

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

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

```
997 \bool_new:N \l_fontspec_external_bool
998 \define@key[zf]{preparse-external}{ExternalLocation}[]{}
999 \zf@nobftrue
1000 \zf@noittrue
1001 \bool_set_true:N \l_fontspec_external_bool
1002 \cs_gset:Npn \fontspec_namewrap:n ##1 { [ #1 ##1 ] }
1003 \xetex_if_engine:T { \setkeys[zf]{preparse}{Renderer=ICU} }
1004 }
1005 \aliasfontfeature{ExternalLocation}{Path}
```

Extension For fonts that aren’t installed in the system. Specifies the font extension to use.

```
1006 \define@key[zf]{preparse-external}{Extension}{
1007   \tl_set:Nn \l_fontspec_extension_tl {#1}
1008   \bool_if:NF \l_fontspec_external_bool {
1009     \setkeys*[zf]{preparse-external}{ExternalLocation}
1010   }
1011 }
1012 \tl_clear:N \l_fontspec_extension_tl
```

22.9.2 Pre-parsed features

After the font name(s) have been sorted out, now need to extract any renderer/font configuration features that need to be processed before all other font features.

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

```
1013 \define@choicekey[zf]{preparse}{Renderer}[\l_tmpa_tl\l_tmpa_num]
1014   {AAT,ICU,Graphite,Full,Basic}{
1015   \zf@update@family{+rend:#1}
1016   \intexpr_compare:nTF {\l_tmpa_num 3} {
1017     \xetex_or luatex:nn
1018     {
```

```

1019 \tl_set:Nv \l_fontspec_renderer_tl {g_fontspec_renderer_tag_\l_tmpa_tl}
1020 }
1021 {
1022 \fontspec_warning:nx {only-xetex-feature} {Renderer=AAT/ICU/Graphite}
1023 }
1024 }{
1025 \xetex_or luatex:nn
1026 { \fontspec_warning:nx {only-luatex-feature} {Renderer=Full/Basic} }
1027 { \tl_set:Nv \l_fontspec_mode_tl {g_fontspec_mode_tag_\l_tmpa_tl} }
1028 }
1029 }
1030 \tl_set:cn {g_fontspec_renderer_tag_AAT} {/AAT}
1031 \tl_set:cn {g_fontspec_renderer_tag_ICU} {/ICU}
1032 \tl_set:cn {g_fontspec_renderer_tag_Graphite} {/GR}
1033 \tl_set:cn {g_fontspec_mode_tag_Full} {node}
1034 \tl_set:cn {g_fontspec_mode_tag_Basic} {base}

```

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

```

1035 \define@key[zf]{preparse}{Script}{
1036 \xetex_if_engine:T { \setkeys[zf]{preparse}{Renderer=ICU} }
1037 \tl_set:Nn \l_fontspec_script_name_tl {#1}
1038 \zf@update@family{+script:#1}
1039 }

```

Exactly the same:

```

1040 \define@key[zf]{preparse}{Language}{
1041 \xetex_if_engine:T { \setkeys[zf]{preparse}{Renderer=ICU} }
1042 \tl_set:Nn \l_fontspec_lang_name_tl {#1}
1043 \zf@update@family{+language:#1}
1044 }

```

22.9.3 Bold/italic choosing options

The Bold, Italic, and BoldItalic features are for defining explicitly the bold and italic fonts used in a font family.

Fonts Upright:

```

1045 \define@key[zf]{preparse-external}{UprightFont}{
1046 \fontspec_complete_fontname:Nn \zf@up {#1}
1047 \zf@update@family{up:#1}
1048 }

```

Bold:

```

1049 \define@key[zf]{preparse-external}{BoldFont}{
1050 \edef\@tempa{#1}
1051 \ifx\@tempa\empty
1052 \zf@nobftrue
1053 \zf@update@family{nobf}
1054 \else
1055 \zf@nobffalse

```

```

1056 \fontspec_complete_fontname:Nn \zf@bf {#1}
1057 \zf@update@family{bf:#1}
1058 \fi
1059 }

```

Same for italic:

```

1060 \define@key[zf]{preparse-external}{ItalicFont}{
1061 \edef\@tempa{#1}
1062 \ifx\@tempa\@empty
1063 \zf@noittrue
1064 \zf@update@family{noit}
1065 \else
1066 \zf@noitfalse
1067 \fontspec_complete_fontname:Nn \zf@it {#1}
1068 \zf@update@family{it:#1}
1069 \fi
1070 }

```

Simpler for bold+italic & slanted:

```

1071 \define@key[zf]{preparse-external}{BoldItalicFont}{
1072 \fontspec_complete_fontname:Nn \zf@bfit {#1}
1073 \zf@update@family{bfit:#1}
1074 }
1075 \define@key[zf]{preparse-external}{SlantedFont}{
1076 \fontspec_complete_fontname:Nn \zf@sl {#1}
1077 \zf@update@family{sl:#1}
1078 }
1079 \define@key[zf]{preparse-external}{BoldSlantedFont}{
1080 \fontspec_complete_fontname:Nn \zf@bfsl {#1}
1081 \zf@update@family{bfsl:#1}
1082 }

```

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

```

1083 \define@key[zf]{options}{SmallCapsFont}{
1084 \edef\@tempa{#1}
1085 \ifx\@tempa\@empty
1086 \zf@nosctrue
1087 \zf@update@family{nosc}
1088 \else
1089 \zf@noscfalse
1090 \fontspec_complete_fontname:Nn \zf@sc {#1}
1091 \zf@update@family{sc:\zap@space #1~\@empty}
1092 \fi
1093 }

```

\fontspec_complete_fontname:Nn This macro defines #1 as the input with any * tokens of its input replaced by the font name. This lets us define supplementary fonts in full ("Baskerville Semibold") or in abbreviation ("* Semibold").

```

1094 \cs_set:Npn \fontspec_complete_fontname:Nn #1#2 {
1095 \tl_set:Nn #1 {#2}
1096 \tl_replace_all_in:Nnx #1 {*} {\zf@fontname}
1097 }
1098 \cs_generate_variant:Nn \tl_replace_all_in:Nnn {Nnx}

```

Features

```
1099 \define@key[zf]{preparse}{UprightFeatures}{
1100   \def\zf@up@feat{, #1}
1101   \zf@update@family{rmfeat:#1}
1102 }
1103 \define@key[zf]{preparse}{BoldFeatures}{
1104   \def\zf@bf@feat{, #1}
1105   \zf@update@family{bf feat:#1}
1106 }
1107 \define@key[zf]{preparse}{ItalicFeatures}{
1108   \def\zf@it@feat{, #1}
1109   \zf@update@family{it feat:#1}
1110 }
1111 \define@key[zf]{preparse}{BoldItalicFeatures}{
1112   \def\zf@bfit@feat{, #1}
1113   \zf@update@family{bfit feat:#1}
1114 }
1115 \define@key[zf]{preparse}{SlantedFeatures}{
1116   \def\zf@sl@feat{, #1}
1117   \zf@update@family{sl feat:#1}
1118 }
1119 \define@key[zf]{preparse}{BoldSlantedFeatures}{
1120   \def\zf@bfs@feat{, #1}
1121   \zf@update@family{bfs@feat:#1}
1122 }
```

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

```
1123 \define@key[zf]{options}{SmallCapsFeatures}{
1124   \unless\ifzf@firsttime\def\zf@sc@feat{, #1}\fi
1125   \zf@update@family{sc feat:\zap@space #1~\@empty}
1126 }
```

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

```
1127 \define@key[zf]{preparse}{SizeFeatures}{
1128   \unless\ifzf@firsttime\def\l_fontspec_sizefeat_clist{#1}\fi
1129   \zf@update@family{sizefeat:\zap@space #1~\@empty}
1130 }
1131 \define@key[zf]{sizing}{Size}{ \tl_set:Nn \l_fontspec_size_tl {#1} }
1132 \define@key[zf]{sizing}{Font}{
1133   \fontspec_complete_fontname:Nn \l_fontspec_sizedfont_tl {#1}
1134 }
```

22.9.4 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. `\fontspec_calc_scale:n` does all the work in the auto-scaling cases.

```
1135 \define@key[zf]{options}{Scale}{
1136   \prg_case_str:nnn {#1}
```



```

1137 {
1138   {MatchLowercase} { \fontspec_calc_scale:n {5} }
1139   {MatchUppercase} { \fontspec_calc_scale:n {8} }
1140 }
1141 { \tl_set:Nx \l_fontspec_scale_tl {#1} }
1142 \zf@update@family{+scale:\l_fontspec_scale_tl}
1143 \tl_set:Nx \l_fontspec_scale_tl { s*[\l_fontspec_scale_tl] }
1144 }

```

`\fontspec_calc_scale:n` This macro calculates the amount of scaling between the default roman font and the (default shape of) the font being selected such that the font dimension that is input is equal for both. The only font dimensions that justify this are 5 (lowercase height) and 8 (uppercase height in $X_{\text{T}}\text{E}^{\text{X}}$).

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

```

1145 \cs_new:Npn \fontspec_calc_scale:n #1 {
1146   \group_begin:
1147   \rmfamily
1148   \fontspec_set_font_dimen:NnN \@tempdima {#1} \font
1149   \fontspec_set_font_dimen:NnN \@tempdimb {#1} \zf@basefont
1150   \dim_set:Nn \@tempdimc { 1pt*\@tempdima/\@tempdimb }
1151   \tl_gset:Nx \l_fontspec_scale_tl {\strip@pt\@tempdimc}
1152   \fontspec_info:n {set-scale}
1153   \group_end:
1154 }

```

This function sets the dimension #1 (for font #3) to what would be known as the ‘fontdimen’ #2 in $\text{T}_{\text{E}}\text{X}$ but apparently isn’t set that way in $\text{LuaT}_{\text{E}}\text{X}$.

```

1155 \xetex_or luatex:nnn { \cs_new:Npn \fontspec_set_font_dimen:NnN #1#2#3 }
1156 {
1157   \dim_set:Nn #1 { \fontdimen #2 #3 }
1158 }
1159 {
1160   \dim_set:Nn #1 { \directlua{ fontspec.get_dimen(#2, "\cs_to_str:N #3") } }
1161 }

```

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

```

1162 \define@key[zf]{options}{WordSpace}{
1163   \zf@update@family{+wordspace:#1}
1164   \unless\ifzf@firsttime
1165     \zf@wordspace@parse#1,\zf@ii,\zf@iii,\zf@
1166     \fi
1167 }

```

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

```

1168 \def\zf@wordspace@parse#1,#2,#3,#4\zf@@{
1169   \def\@tempa{#4}

```

```

1170 \ifx\@tempa\@empty
1171   \setlength\@tempdima{#1\fontdimen2\zf@basefont}
1172   \@tempdimb\@tempdima
1173   \@tempdimc\@tempdima
1174 \else
1175   \setlength\@tempdima{#1\fontdimen2\zf@basefont}
1176   \setlength\@tempdimb{#2\fontdimen3\zf@basefont}
1177   \setlength\@tempdimc{#3\fontdimen4\zf@basefont}
1178 \fi
1179 \edef\zf@adjust{
1180   \zf@adjust
1181   \fontdimen2\font\the\@tempdima
1182   \fontdimen3\font\the\@tempdimb
1183   \fontdimen4\font\the\@tempdimc
1184 }
1185 }

```

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

```

1186 \define@key[zf]{options}{PunctuationSpace}{
1187   \zf@update@family{+punctspace:#1}
1188   \setlength\@tempdima{#1\fontdimen7\zf@basefont}
1189   \edef\zf@adjust{\zf@adjust\fontdimen7\font\the\@tempdima}
1190 }

```

Letterspacing

```

1191 \define@key[zf]{options}{LetterSpace}{
1192   \zf@update@family{+tracking:#1}
1193   \zf@update@ff{letterspace=#1}
1194 }

```

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.

```

1195 \define@key[zf]{options}{HyphenChar}{
1196   \zf@update@family{+hyphenchar:#1}
1197   \edef\@tempa{#1}
1198   \edef\@tempb{None}
1199   \ifx\@tempa\@tempb
1200     \xetex_or luatex:nnn { \g@addto@macro\zf@adjust }
1201     { \hyphenchar\font-1\relax }
1202     { \luatexprehyphenchar=-1\relax }
1203 \else
1204   \zf@check@one@char#1\zf@@
1205   \ifx\@tempb\@empty
1206     \xetex_or luatex:nn {
1207       {\zf@basefont\expandafter\ifnum\expandafter\XeTeXcharglyph
1208         \expandafter‘#1 \z@
1209         \g@addto@macro\zf@adjust{%
1210           {\expandafter\hyphenchar\expandafter
1211             \font\expandafter‘#1}}%

```

```

1212     \else
1213     \fontspec_error:nx {no-glyph}{#1}
1214     \fi}%
1215   }{
1216     \ifnum\directlua{
1217       fontspec.sprint(fontspec.charglyph('#1', 'zf@basefont'))
1218     } \z@
1219     \g@addto@macro\zf@adjust{\luatexprehyphenchar='#1\relax}
1220     \else
1221     \fontspec_error:nx {no-glyph}{#1}
1222     \fi
1223   }
1224 \else
1225   \xetex_or luatex:nn {
1226     {\zf@basefont\ifnum\XeTeXcharglyph#1 \z@
1227       \g@addto@macro\zf@adjust{\hyphenchar\font#1\relax}%
1228     \else
1229       \fontspec_error:nx {no-glyph}{#1}
1230     \fi}%
1231   }{
1232     \ifnum\directlua{
1233       fontspec.sprint(fontspec.charglyph('#1', 'zf@basefont'))
1234     } \z@
1235     \g@addto@macro\zf@adjust{\luatexprehyphenchar=#1\relax}
1236     \else
1237     \fontspec_error:nx {no-glyph}{#1}
1238     \fi
1239   }
1240 \fi
1241 \fi
1242 }
1243 \def\zf@check@one@char#1#2\zf@@{\def\@tempb{#2}}

```

Color

```

1244 \define@key[zf]{options}{Color}{
1245   \zf@update@family{+col:#1}
1246   \zf@update@ff{color=#1}
1247 }
1248 \keyval@alias@key[zf]{options}{Color}{Colour}

```

Mapping

```

1249 \xetex_or luatex:nnn {
1250   \define@key[zf]{options}{Mapping}
1251 }{
1252   \zf@update@family{+map:#1}
1253   \zf@update@ff{mapping=#1}
1254 }{
1255   \tl_if_eq:nnTF {#1} {tex-text} {
1256     \fontspec_warning:n {no-mapping-ligtext}
1257     \msg_redirect_name:nnn {fontspec} {no-mapping-ligtext} {none}

```

```

1258 \setkeys[zf]{options}{ Ligatures=TeX }
1259 }{
1260 \fontspec_warning:n {no-mapping}
1261 }
1262 }

```

FeatureFile

```

1263 \define@key[zf]{options}{FeatureFile}{
1264 \zf@update@family{+fea:#1}
1265 \zf@update@ff{featurefile=#1}
1266 }

```

22.9.5 Continuous font axes

```

1267 \define@key[zf]{options}{Weight}{
1268 \zf@update@family{+weight:#1}
1269 \zf@update@ff{weight=#1}
1270 }
1271 \define@key[zf]{options}{Width}{
1272 \zf@update@family{+width:#1}
1273 \zf@update@ff{width=#1}
1274 }
1275 \define@key[zf]{options}{OpticalSize}{
1276 \xetex_or luatex:nn {
1277 \ifzf@icu
1278 \tl_set:Nn \l_fontspec_optical_size_tl {/ S = #1}
1279 \zf@update@family{+size:#1}
1280 \fi
1281 \ifzf@mm
1282 \zf@update@family{+size:#1}
1283 \zf@update@ff{optical size=#1}
1284 \fi
1285 \ifzf@icu\else
1286 \ifzf@mm\else
1287 \ifzf@firsttime
1288 \fontspec_warning:n {no-opticals}
1289 \fi
1290 \fi
1291 \fi
1292 }{
1293 \tl_set:Nn \l_fontspec_optical_size_tl {/ S = #1}
1294 \zf@update@family{+size:#1}
1295 }
1296 }

```

22.9.6 Font transformations

```

1297 \define@key[zf]{options}{FakeSlant}{
1298 \zf@update@family{+slant:#1}
1299 \zf@update@ff{slant=#1}
1300 }
1301 \define@key[zf]{options}{FakeStretch}{

```

```

1302 \zf@update@family{+extend:#1}
1303 \zf@update@ff{extend=#1}
1304 }
1305 \define@key[zf]{options}{FakeBold}{
1306 \zf@update@family{+embolden:#1}
1307 \zf@update@ff{embolden=#1}
1308 }

```

22.9.7 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.

```

1309 \zf@define@font@feature{Ligatures}
1310 \zf@define@feature@option{Ligatures}{Required}      {1}{0}{+rlig}
1311 \zf@define@feature@option{Ligatures}{NoRequired}   {1}{1}{-rlig}
1312 \zf@define@feature@option{Ligatures}{Common}      {1}{2}{+liga}
1313 \zf@define@feature@option{Ligatures}{NoCommon}    {1}{3}{-liga}
1314 \zf@define@feature@option{Ligatures}{Rare}        {1}{4}{+dlig}
1315 \zf@define@feature@option{Ligatures}{NoRare}      {1}{5}{-dlig}
1316 \zf@define@feature@option{Ligatures}{Discretionary} {1}{4}{+dlig}
1317 \zf@define@feature@option{Ligatures}{NoDiscretionary} {1}{5}{-dlig}
1318 \zf@define@feature@option{Ligatures}{Contextual}   {}{} {+clig}
1319 \zf@define@feature@option{Ligatures}{NoContextual} {}{} {-clig}
1320 \zf@define@feature@option{Ligatures}{Historical}   {}{} {+hlig}
1321 \zf@define@feature@option{Ligatures}{NoHistorical} {}{} {-hlig}
1322 \zf@define@feature@option{Ligatures}{Logos}       {1}{6} {}
1323 \zf@define@feature@option{Ligatures}{NoLogos}     {1}{7} {}
1324 \zf@define@feature@option{Ligatures}{Rebus}       {1}{8} {}
1325 \zf@define@feature@option{Ligatures}{NoRebus}    {1}{9} {}
1326 \zf@define@feature@option{Ligatures}{Diphthong}  {1}{10} {}
1327 \zf@define@feature@option{Ligatures}{NoDiphthong} {1}{11} {}
1328 \zf@define@feature@option{Ligatures}{Squared}    {1}{12} {}
1329 \zf@define@feature@option{Ligatures}{NoSquared}  {1}{13} {}
1330 \zf@define@feature@option{Ligatures}{AbbrevSquared} {1}{14} {}
1331 \zf@define@feature@option{Ligatures}{NoAbbrevSquared} {1}{15} {}
1332 \zf@define@feature@option{Ligatures}{Icelandic}  {1}{32} {}
1333 \zf@define@feature@option{Ligatures}{NoIcelandic} {1}{33} {}

```

Emulate CM extra ligatures.

```

1334 \define@key[zf@feat]{Ligatures}{TeX}[]{}
1335 \xetex_or luatex:nn {
1336 \zf@update@family{+map:tex-text}
1337 \zf@update@ff{mapping=tex-text}
1338 }{
1339 \zf@update@family{+tlig+trep}
1340 \zf@update@ff{+tlig;+trep}
1341 }
1342 }

```

22.9.8 Letters

```

1343 \zf@define@font@feature{Letters}
1344 \zf@define@feature@option{Letters}{Normal} {3}{0}{}
1345 \zf@define@feature@option{Letters}{Uppercase} {3}{1}{+case}
1346 \zf@define@feature@option{Letters}{Lowercase} {3}{2}{}
1347 \zf@define@feature@option{Letters}{SmallCaps} {3}{3}{+smcp}
1348 \zf@define@feature@option{Letters}{PetiteCaps} {} {} {+pcap}
1349 \zf@define@feature@option{Letters}{UppercaseSmallCaps} {} {} {+c2sc}
1350 \zf@define@feature@option{Letters}{UppercasePetiteCaps} {} {} {+c2pc}
1351 \zf@define@feature@option{Letters}{InitialCaps} {3}{4}{}
1352 \zf@define@feature@option{Letters}{Unicase} {} {} {+unic}

```

22.9.9 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.

```

1353 \zf@define@font@feature{Numbers}
1354 \zf@define@feature@option{Numbers}{Monospaced} {6}{0}{+tnum}
1355 \zf@define@feature@option{Numbers}{Proportional} {6}{1}{+pnum}
1356 \zf@define@feature@option{Numbers}{Lowercase} {21}{0}{+onum}
1357 \zf@define@feature@option{Numbers}{OldStyle} {21}{0}{+onum}
1358 \zf@define@feature@option{Numbers}{Uppercase} {21}{1}{+lnum}
1359 \zf@define@feature@option{Numbers}{Lining} {21}{1}{+lnum}
1360 \zf@define@feature@option{Numbers}{SlashedZero} {14}{5}{+zero}
1361 \zf@define@feature@option{Numbers}{NoSlashedZero} {14}{4}{-zero}

```

luaotload provides a custom anum feature for replacing Latin (AKA Arabic) numbers with Arabic (AKA Indic-Arabic). The same feature maps to Farsi (Persian) numbers if font language is Farsi.

```

1362 \luatex_if_engine:T {
1363   \zf@define@feature@option{Numbers}{Arabic}{}{}{+anum}
1364   \zf@define@feature@option{Numbers}{Farsi}{}{}{+anum}
1365 }

```

22.9.10 Contextuals

```

1366 \zf@define@font@feature {Contextuals}
1367 \zf@define@feature@option{Contextuals}{Swash} {} {} {+csw}
1368 \zf@define@feature@option{Contextuals}{NoSwash} {} {} {-csw}
1369 \zf@define@feature@option{Contextuals}{Alternate} {} {} {+calt}
1370 \zf@define@feature@option{Contextuals}{NoAlternate} {} {} {-calt}
1371 \zf@define@feature@option{Contextuals}{WordInitial} {8}{0}{+init}
1372 \zf@define@feature@option{Contextuals}{NoWordInitial} {8}{1}{-init}
1373 \zf@define@feature@option{Contextuals}{WordFinal} {8}{2}{+fina}
1374 \zf@define@feature@option{Contextuals}{NoWordFinal} {8}{3}{-fina}
1375 \zf@define@feature@option{Contextuals}{LineInitial} {8}{4}{}
1376 \zf@define@feature@option{Contextuals}{NoLineInitial} {8}{5}{}
1377 \zf@define@feature@option{Contextuals}{LineFinal} {8}{6}{+falt}
1378 \zf@define@feature@option{Contextuals}{NoLineFinal} {8}{7}{-falt}
1379 \zf@define@feature@option{Contextuals}{Inner} {8}{8}{+medi}
1380 \zf@define@feature@option{Contextuals}{NoInner} {8}{9}{-medi}

```

22.9.11 Diacritics

```
1381 \zf@define@font@feature{Diacritics}
1382 \zf@define@feature@option{Diacritics}{Show}      {9}{0}{ }
1383 \zf@define@feature@option{Diacritics}{Hide}      {9}{1}{ }
1384 \zf@define@feature@option{Diacritics}{Decompose} {9}{2}{ }
1385 \zf@define@feature@option{Diacritics}{MarkToBase} { }{ }{+mark}
1386 \zf@define@feature@option{Diacritics}{NoMarkToBase} { }{ }{-mark}
1387 \zf@define@feature@option{Diacritics}{MarkToMark} { }{ }{+mkmk}
1388 \zf@define@feature@option{Diacritics}{NoMarkToMark} { }{ }{-mkmk}
1389 \zf@define@feature@option{Diacritics}{AboveBase}  { }{ }{+abvm}
1390 \zf@define@feature@option{Diacritics}{NoAboveBase} { }{ }{-abvm}
1391 \zf@define@feature@option{Diacritics}{BelowBase}  { }{ }{+blwm}
1392 \zf@define@feature@option{Diacritics}{NoBelowBase} { }{ }{-blwm}
```

22.9.12 Kerning

```
1393 \zf@define@font@feature{Kerning}
1394 \zf@define@feature@option{Kerning}{Uppercase} { }{ }{+csp}
1395 \zf@define@feature@option{Kerning}{On}        { }{ }{+kern}
1396 \zf@define@feature@option{Kerning}{Off}       { }{ }{-kern}
1397 %\zf@define@feature@option{Kerning}{Vertical} { }{ }{+vkern}
1398 %\zf@define@feature@option{Kerning}
1399 %      {VerticalAlternateProportional} { }{ }{+vpal}
1400 %\zf@define@feature@option{Kerning}{VerticalAlternateHalfWidth} { }{ }{+vhal}
```

22.9.13 Vertical position

```
1401 \zf@define@font@feature{VerticalPosition}
1402 \zf@define@feature@option{VerticalPosition}{Normal} {10}{0}{ }
1403 \zf@define@feature@option{VerticalPosition}{Superior} {10}{1}{+sup}
1404 \zf@define@feature@option{VerticalPosition}{Inferior} {10}{2}{+sub}
1405 \zf@define@feature@option{VerticalPosition}{Ordinal} {10}{3}{+ordn}
1406 \zf@define@feature@option{VerticalPosition}{Numerator} { } { } {+numr}
1407 \zf@define@feature@option{VerticalPosition}{Denominator} { } { } {+dnom}
1408 \zf@define@feature@option{VerticalPosition}{ScientificInferior} { }{ }{+sinf}
```

22.9.14 Fractions

```
1409 \zf@define@font@feature{Fractions}
1410 \zf@define@feature@option{Fractions}{On}      {11}{1}{+frac}
1411 \zf@define@feature@option{Fractions}{Off}     {11}{0}{-frac}
1412 \zf@define@feature@option{Fractions}{Diagonal} {11}{2}{ }
1413 \zf@define@feature@option{Fractions}{Alternate} { } { } {+afrc}
```

22.9.15 Alternates and variants

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

```
1414 \define@key[zf]{options}{Alternate}[0]{
1415   \setkeys*{zf@feat}{Alternate}{#1}
1416   \unless\ifx\XKV@rm@empty
1417     \def\XKV@tfam{Alternate}
1418     \fontspec_make_feature:nnn{17}{#1}{+salt=#1}
1419   \fi
```

```

1420 }
1421 \define@key[zf]{options}{Variant}{
1422   \setkeys*[zf@feat]{Variant}{#1}
1423   \unless\ifx\XKV@rm\@empty
1424     \def\XKV@tfam{Variant}
1425     \fontspec_make_feature:nnn{18}{#1}{+ss\two@digits{#1}}
1426   \fi
1427 }
1428 \aliasfontfeature{Variant}{StylisticSet}

```

22.9.16 Style

```

1429 \zf@define@font@feature{Style}
1430 \zf@define@feature@option{Style}{Alternate}      {} {} {+salt}
1431 \zf@define@feature@option{Style}{Italic}         {32}{2}{+ital}
1432 \zf@define@feature@option{Style}{Ruby}          {28}{2}{+ruby}
1433 \zf@define@feature@option{Style}{Swash}         {} {} {+swsh}
1434 \zf@define@feature@option{Style}{Historic}      {} {} {+hist}
1435 \zf@define@feature@option{Style}{Display}       {19}{1}{}
1436 \zf@define@feature@option{Style}{Engraved}      {19}{2}{}
1437 \zf@define@feature@option{Style}{TitlingCaps}   {19}{4}{+titl}
1438 \zf@define@feature@option{Style}{TallCaps}      {19}{5}{}
1439 \zf@define@feature@option{Style}{HorizontalKana}{} {} {+hkna}
1440 \zf@define@feature@option{Style}{VerticalKana}  {} {} {+vkna}

```

22.9.17 CJK shape

```

1441 \zf@define@font@feature{CJKShape}
1442 \zf@define@feature@option{CJKShape}{Traditional}{20}{0} {+trad}
1443 \zf@define@feature@option{CJKShape}{Simplified} {20}{1} {+smp1}
1444 \zf@define@feature@option{CJKShape}{JIS1978}   {20}{2} {+jp78}
1445 \zf@define@feature@option{CJKShape}{JIS1983}   {20}{3} {+jp83}
1446 \zf@define@feature@option{CJKShape}{JIS1990}   {20}{4} {+jp90}
1447 \zf@define@feature@option{CJKShape}{Expert}    {20}{10}{+expt}
1448 \zf@define@feature@option{CJKShape}{NLC}       {20}{13}{+nlck}

```

22.9.18 Character width

```

1449 \zf@define@font@feature{CharacterWidth}
1450 \zf@define@feature@option{CharacterWidth}{Proportional}{22}{0}{+pwid}
1451 \zf@define@feature@option{CharacterWidth}{Full}{22}{1}{+fwid}
1452 \zf@define@feature@option{CharacterWidth}{Half}{22}{2}{+hwid}
1453 \zf@define@feature@option{CharacterWidth}{Third}{22}{3}{+twid}
1454 \zf@define@feature@option{CharacterWidth}{Quarter}{22}{4}{+qwid}
1455 \zf@define@feature@option{CharacterWidth}{AlternateProportional}{22}{5}{+palt}
1456 \zf@define@feature@option{CharacterWidth}{AlternateHalf}{22}{6}{+halt}
1457 \zf@define@feature@option{CharacterWidth}{Default}{22}{7}{}

```

22.9.19 Annotation

```

1458 \zf@define@font@feature{Annotation}
1459 \zf@define@feature@option{Annotation}{Off}{24}{0}{-nalt}
1460 \zf@define@feature@option{Annotation}{On}{}{}{+nalt}
1461 \zf@define@feature@option{Annotation}{Box}{24}{1}{}

```



```

1462 \zf@define@feature@option{Annotation}{RoundedBox}{24}{2}{}
1463 \zf@define@feature@option{Annotation}{Circle}{24}{3}{}
1464 \zf@define@feature@option{Annotation}{BlackCircle}{24}{4}{}
1465 \zf@define@feature@option{Annotation}{Parenthesis}{24}{5}{}
1466 \zf@define@feature@option{Annotation}{Period}{24}{6}{}
1467 \zf@define@feature@option{Annotation}{RomanNumerals}{24}{7}{}
1468 \zf@define@feature@option{Annotation}{Diamond}{24}{8}{}
1469 \zf@define@feature@option{Annotation}{BlackSquare}{24}{9}{}
1470 \zf@define@feature@option{Annotation}{BlackRoundSquare}{24}{10}{}
1471 \zf@define@feature@option{Annotation}{DoubleCircle}{24}{11}{}

```

22.9.20 Vertical

```

1472 \zf@define@font@feature{Vertical}
1473 \define@key{zf@feat}{Vertical}{RotatedGlyphs}[] {
1474   \ifzf@icu
1475     \fontspec_make_feature:nnn{}{}{+vrt2}
1476     \zf@update@family{+vert}
1477     \zf@update@ff{vertical}
1478   \else
1479     \zf@update@family{+vert}
1480     \zf@update@ff{vertical}
1481   \fi
1482 }

```

22.9.21 Script

```

1483 \newfontscript{Arabic}{arab}           \newfontscript{Armenian}{armn}
1484 \newfontscript{Balinese}{bali}         \newfontscript{Bengali}{beng}
1485 \newfontscript{Bopomofo}{bopo}        \newfontscript{Braille}{brai}
1486 \newfontscript{Buginese}{bugi}        \newfontscript{Buhid}{buhd}
1487 \newfontscript{Byzantine~Music}{byzm}  \newfontscript{Canadian~Syllabics}{cans}
1488 \newfontscript{Cherokee}{cher}
1489 \newfontscript{CJK~Ideographic}{hani}  \newfontscript{Coptic}{copt}
1490 \newfontscript{Cypriot~Syllabary}{cpri} \newfontscript{Cyrillic}{cyr}
1491 \newfontscript{Default}{DFLT}          \newfontscript{Deseret}{dsrt}
1492 \newfontscript{Devanagari}{deva}        \newfontscript{Ethiopic}{ethi}
1493 \newfontscript{Georgian}{geor}         \newfontscript{Glagolitic}{glag}
1494 \newfontscript{Gothic}{goth}           \newfontscript{Greek}{grek}
1495 \newfontscript{Gujarati}{gujr}        \newfontscript{Gurmukhi}{guru}
1496 \newfontscript{Hangul~Jamo}{jamo}      \newfontscript{Hangul}{hang}
1497 \newfontscript{Hanunoo}{hano}          \newfontscript{Hebrew}{hebr}
1498 \newfontscript{Hiragana~and~Katakana}{kana}
1499 \newfontscript{Javanese}{java}         \newfontscript{Kannada}{knda}
1500 \newfontscript{Kharosthi}{khar}       \newfontscript{Khmer}{khmr}
1501 \newfontscript{Lao}{lao~}             \newfontscript{Latin}{latn}
1502 \newfontscript{Limbu}{limb}           \newfontscript{Linear~B}{linb}
1503 \newfontscript{Malayalam}{mlym}       \newfontscript{Math}{math}
1504 \newfontscript{Mongolian}{mong}
1505 \newfontscript{Musical~Symbols}{musc}  \newfontscript{Myanmar}{mymr}
1506 \newfontscript{N'ko}{nko~}           \newfontscript{Ogham}{ogam}
1507 \newfontscript{Old~Italic}{ital}
1508 \newfontscript{Old~Persian~Cuneiform}{xpeo}
1509 \newfontscript{Oriya}{orya}           \newfontscript{Osmanya}{osma}

```

```

1510 \newfontscript{Phags-pa}{phag}          \newfontscript{Phoenician}{phnx}
1511 \newfontscript{Runic}{runr}             \newfontscript{Shavian}{shaw}
1512 \newfontscript{Sinhala}{sinh}
1513 \newfontscript{Sumero-Akkadian~Cuneiform}{xsux}
1514 \newfontscript{Syloti~Nagri}{sylo}      \newfontscript{Syriac}{syr}
1515 \newfontscript{Tagalog}{tglg}           \newfontscript{Tagbanwa}{tagb}
1516 \newfontscript{Tai~Le}{tale}           \newfontscript{Tai~Lu}{tal}
1517 \newfontscript{Tamil}{taml}             \newfontscript{Telugu}{telu}
1518 \newfontscript{Thaana}{thaa}           \newfontscript{Thai}{thai}
1519 \newfontscript{Tibetan}{tib}           \newfontscript{Tifinagh}{tfn}
1520 \newfontscript{Ugaritic~Cuneiform}{ugar}\newfontscript{Yi}{yi}

```

For convenience:

```

1521 \newfontscript{Kana}{kana}
1522 \newfontscript{Maths}{math}
1523 \newfontscript{CJK}{hani}

```

22.9.22 Language

```

1524 \newfontlanguage{Abaza}{ABA}\newfontlanguage{Abkhazian}{ABK}
1525 \newfontlanguage{Adyghe}{ADY}\newfontlanguage{Afrikaans}{AFK}
1526 \newfontlanguage{Afar}{AFR}\newfontlanguage{Agaw}{AGW}
1527 \newfontlanguage{Altai}{ALT}\newfontlanguage{Amharic}{AMH}
1528 \newfontlanguage{Arabic}{ARA}\newfontlanguage{Aari}{ARI}
1529 \newfontlanguage{Arakanese}{ARK}\newfontlanguage{Assamese}{ASM}
1530 \newfontlanguage{Athapaskan}{ATH}\newfontlanguage{Avar}{AVR}
1531 \newfontlanguage{Awadhi}{AWA}\newfontlanguage{Aymara}{AYM}
1532 \newfontlanguage{Azeri}{AZE}\newfontlanguage{Badaga}{BAD}
1533 \newfontlanguage{Baghelkhandi}{BAG}\newfontlanguage{Balkar}{BAL}
1534 \newfontlanguage{Baule}{BAU}\newfontlanguage{Berber}{BBR}
1535 \newfontlanguage{Bench}{BCH}\newfontlanguage{Bible~Cree}{BCR}
1536 \newfontlanguage{Belarussian}{BEL}\newfontlanguage{Bemba}{BEM}
1537 \newfontlanguage{Bengali}{BEN}\newfontlanguage{Bulgarian}{BGR}
1538 \newfontlanguage{Bhili}{BHI}\newfontlanguage{Bhojpuri}{BHO}
1539 \newfontlanguage{Bikol}{BIK}\newfontlanguage{Bilen}{BIL}
1540 \newfontlanguage{Blackfoot}{BKF}\newfontlanguage{Balochi}{BLI}
1541 \newfontlanguage{Balante}{BLN}\newfontlanguage{Balti}{BLT}
1542 \newfontlanguage{Bambara}{BMB}\newfontlanguage{Bamileke}{BML}
1543 \newfontlanguage{Breton}{BRE}\newfontlanguage{Brahui}{BRH}
1544 \newfontlanguage{Braj~Bhasha}{BRI}\newfontlanguage{Burmese}{BRM}
1545 \newfontlanguage{Bashkir}{BSH}\newfontlanguage{Beti}{BTI}
1546 \newfontlanguage{Catalan}{CAT}\newfontlanguage{Cebuano}{CEB}
1547 \newfontlanguage{Chechen}{CHE}\newfontlanguage{Chaha~Gurage}{CHG}
1548 \newfontlanguage{Chattisgarhi}{CHH}\newfontlanguage{Chichewa}{CHI}
1549 \newfontlanguage{Chukchi}{CHK}\newfontlanguage{Chipewyan}{CHP}
1550 \newfontlanguage{Cherokee}{CHR}\newfontlanguage{Chuvash}{CHU}
1551 \newfontlanguage{Comorian}{CMR}\newfontlanguage{Coptic}{COP}
1552 \newfontlanguage{Cree}{CRE}\newfontlanguage{Carrier}{CRR}
1553 \newfontlanguage{Crimean~Tatar}{CRT}\newfontlanguage{Church~Slavonic}{CSL}
1554 \newfontlanguage{Czech}{CSY}\newfontlanguage{Danish}{DAN}
1555 \newfontlanguage{Dargwa}{DAR}\newfontlanguage{Woods~Cree}{DCR}
1556 \newfontlanguage{German}{DEU}
1557 \newfontlanguage{Dogri}{DGR}\newfontlanguage{Divehi}{DIV}

```

1558 \newfontlanguage{Djerma}{DJR}\newfontlanguage{Dangme}{DNG}
1559 \newfontlanguage{Dinka}{DNK}\newfontlanguage{Dungan}{DUN}
1560 \newfontlanguage{Dzongkha}{DZN}\newfontlanguage{Ebira}{EBI}
1561 \newfontlanguage{Eastern~Cree}{ECR}\newfontlanguage{Edo}{EDO}
1562 \newfontlanguage{Efik}{EFI}\newfontlanguage{Greek}{ELL}
1563 \newfontlanguage{English}{ENG}\newfontlanguage{Erzya}{ERZ}
1564 \newfontlanguage{Spanish}{ESP}\newfontlanguage{Estonian}{ETI}
1565 \newfontlanguage{Basque}{EUQ}\newfontlanguage{Evenki}{EVK}
1566 \newfontlanguage{Even}{EVN}\newfontlanguage{Ewe}{EWE}
1567 \newfontlanguage{French~Antillean}{FAN}\newfontlanguage{Farsi}{FAR}
1568 \newfontlanguage{Finnish}{FIN}\newfontlanguage{Fijian}{FJI}
1569 \newfontlanguage{Flemish}{FLE}\newfontlanguage{Forest~Nenets}{FNE}
1570 \newfontlanguage{Fon}{FON}\newfontlanguage{Faroese}{FOS}
1571 \newfontlanguage{French}{FRA}\newfontlanguage{Frisian}{FRI}
1572 \newfontlanguage{Friulian}{FRL}\newfontlanguage{Futa}{FTA}
1573 \newfontlanguage{Fulani}{FUL}\newfontlanguage{Ga}{GAD}
1574 \newfontlanguage{Gaelic}{GAE}\newfontlanguage{Gagauz}{GAG}
1575 \newfontlanguage{Galician}{GAL}\newfontlanguage{Garshuni}{GAR}
1576 \newfontlanguage{Garhwali}{GAW}\newfontlanguage{Ge'ez}{GEZ}
1577 \newfontlanguage{Gilyak}{GIL}\newfontlanguage{Gumuz}{GMZ}
1578 \newfontlanguage{Gondi}{GON}\newfontlanguage{Greenlandic}{GRN}
1579 \newfontlanguage{Garo}{GRO}\newfontlanguage{Guarani}{GUA}
1580 \newfontlanguage{Gujarati}{GUJ}\newfontlanguage{Haitian}{HAI}
1581 \newfontlanguage{Halam}{HAL}\newfontlanguage{Harauti}{HAR}
1582 \newfontlanguage{Hausa}{HAU}\newfontlanguage{Hawaiin}{HAW}
1583 \newfontlanguage{Hammer-Banna}{HBN}\newfontlanguage{Hiligaynon}{HIL}
1584 \newfontlanguage{Hindi}{HIN}\newfontlanguage{High~Mari}{HMA}
1585 \newfontlanguage{Hindko}{HND}\newfontlanguage{Ho}{HO}
1586 \newfontlanguage{Harari}{HRI}\newfontlanguage{Croatian}{HRV}
1587 \newfontlanguage{Hungarian}{HUN}\newfontlanguage{Armenian}{HYE}
1588 \newfontlanguage{Igbo}{IBO}\newfontlanguage{Ijo}{IJO}
1589 \newfontlanguage{Ilokano}{ILO}\newfontlanguage{Indonesian}{IND}
1590 \newfontlanguage{Ingush}{ING}\newfontlanguage{Inuktitut}{INU}
1591 \newfontlanguage{Irish}{IRI}\newfontlanguage{Irish~Traditional}{IRT}
1592 \newfontlanguage{Icelandic}{ISL}\newfontlanguage{Inari~Sami}{ISM}
1593 \newfontlanguage{Italian}{ITA}\newfontlanguage{Hebrew}{IWR}
1594 \newfontlanguage{Javanese}{JAV}\newfontlanguage{Yiddish}{JII}
1595 \newfontlanguage{Japanese}{JAN}\newfontlanguage{Judezmo}{JUD}
1596 \newfontlanguage{Jula}{JUL}\newfontlanguage{Kabardian}{KAB}
1597 \newfontlanguage{Kachchi}{KAC}\newfontlanguage{Kalenjin}{KAL}
1598 \newfontlanguage{Kannada}{KAN}\newfontlanguage{Karachay}{KAR}
1599 \newfontlanguage{Georgian}{KAT}\newfontlanguage{Kazakh}{KAZ}
1600 \newfontlanguage{Kebena}{KEB}\newfontlanguage{Khutsuri~Georgian}{KGE}
1601 \newfontlanguage{Khakass}{KHA}\newfontlanguage{Khanty-Kazim}{KHK}
1602 \newfontlanguage{Khmer}{KHM}\newfontlanguage{Khanty-Shurishkar}{KHS}
1603 \newfontlanguage{Khanty-Vakhi}{KHV}\newfontlanguage{Khowar}{KHW}
1604 \newfontlanguage{Kikuyu}{KIK}\newfontlanguage{Kirghiz}{KIR}
1605 \newfontlanguage{Kisii}{KIS}\newfontlanguage{Kokni}{KKN}
1606 \newfontlanguage{Kalmyk}{KLM}\newfontlanguage{Kamba}{KMB}
1607 \newfontlanguage{Kumaoni}{KMN}\newfontlanguage{Komo}{KMO}
1608 \newfontlanguage{Komso}{KMS}\newfontlanguage{Kanuri}{KNR}

1609 \newfontlanguage{Kodagu}{KOD}\newfontlanguage{Korean~Old~Hangul}{KOH}
1610 \newfontlanguage{Konkani}{KOK}\newfontlanguage{Kikongo}{KON}
1611 \newfontlanguage{Komi-Permyak}{KOP}\newfontlanguage{Korean}{KOR}
1612 \newfontlanguage{Komi-Zyrian}{KOZ}\newfontlanguage{Kpelle}{KPL}
1613 \newfontlanguage{Krio}{KRI}\newfontlanguage{Karakalpak}{KPK}
1614 \newfontlanguage{Karelian}{KRL}\newfontlanguage{Karaim}{KRM}
1615 \newfontlanguage{Karen}{KRN}\newfontlanguage{Korete}{KRT}
1616 \newfontlanguage{Kashmiri}{KSH}\newfontlanguage{Khasi}{KSI}
1617 \newfontlanguage{Kildin~Sami}{KSM}\newfontlanguage{Kui}{KUI}
1618 \newfontlanguage{Kulvi}{KUL}\newfontlanguage{Kumyk}{KUM}
1619 \newfontlanguage{Kurdish}{KUR}\newfontlanguage{Kurukh}{KUU}
1620 \newfontlanguage{Kuy}{KUY}\newfontlanguage{Koryak}{KYK}
1621 \newfontlanguage{Ladin}{LAD}\newfontlanguage{Lahuli}{LAH}
1622 \newfontlanguage{Lak}{LAK}\newfontlanguage{Lambani}{LAM}
1623 \newfontlanguage{Lao}{LAO}\newfontlanguage{Latin}{LAT}
1624 \newfontlanguage{Laz}{LAZ}\newfontlanguage{L-Cree}{LCR}
1625 \newfontlanguage{Ladakhi}{LDK}\newfontlanguage{Lezgi}{LEZ}
1626 \newfontlanguage{Lingala}{LIN}\newfontlanguage{Low~Mari}{LMA}
1627 \newfontlanguage{Limbu}{LMB}\newfontlanguage{Lomwe}{LMW}
1628 \newfontlanguage{Lower~Sorbian}{LSB}\newfontlanguage{Lule~Sami}{LSM}
1629 \newfontlanguage{Lithuanian}{LTH}\newfontlanguage{Luba}{LUB}
1630 \newfontlanguage{Luganda}{LUG}\newfontlanguage{Luhya}{LUH}
1631 \newfontlanguage{Luo}{LUO}\newfontlanguage{Latvian}{LVI}
1632 \newfontlanguage{Majang}{MAJ}\newfontlanguage{Makua}{MAK}
1633 \newfontlanguage{Malayalam~Traditional}{MAL}\newfontlanguage{Mansi}{MAN}
1634 \newfontlanguage{Marathi}{MAR}\newfontlanguage{Marwari}{MAW}
1635 \newfontlanguage{Mbundu}{MBN}\newfontlanguage{Manchu}{MCH}
1636 \newfontlanguage{Moose~Cree}{MCR}\newfontlanguage{Mende}{MDE}
1637 \newfontlanguage{Me'en}{MEN}\newfontlanguage{Mizo}{MIZ}
1638 \newfontlanguage{Macedonian}{MKD}\newfontlanguage{Male}{MLE}
1639 \newfontlanguage{Malagasy}{MLG}\newfontlanguage{Malinke}{MLN}
1640 \newfontlanguage{Malayalam~Reformed}{MLR}\newfontlanguage{Malay}{MLY}
1641 \newfontlanguage{Mandinka}{MND}\newfontlanguage{Mongolian}{MNG}
1642 \newfontlanguage{Manipuri}{MNI}\newfontlanguage{Maninka}{MNK}
1643 \newfontlanguage{Manx~Gaelic}{MNX}\newfontlanguage{Moksha}{MOK}
1644 \newfontlanguage{Moldavian}{MOL}\newfontlanguage{Mon}{MON}
1645 \newfontlanguage{Moroccan}{MOR}\newfontlanguage{Maori}{MRI}
1646 \newfontlanguage{Maithili}{MTH}\newfontlanguage{Maltese}{MTS}
1647 \newfontlanguage{Mundari}{MUN}\newfontlanguage{Naga-Assamese}{NAG}
1648 \newfontlanguage{Nanai}{NAN}\newfontlanguage{Naskapi}{NAS}
1649 \newfontlanguage{N-Cree}{NCR}\newfontlanguage{Ndebele}{NDB}
1650 \newfontlanguage{Ndonga}{NDG}\newfontlanguage{Nepali}{NEP}
1651 \newfontlanguage{Newari}{NEW}\newfontlanguage{Nagari}{NGR}
1652 \newfontlanguage{Norway~House~Cree}{NHC}\newfontlanguage{Nisi}{NIS}
1653 \newfontlanguage{Niuean}{NIU}\newfontlanguage{Nkole}{NKL}
1654 \newfontlanguage{N'ko}{NKO}\newfontlanguage{Dutch}{NLD}
1655 \newfontlanguage{Nogai}{NOG}\newfontlanguage{Norwegian}{NOR}
1656 \newfontlanguage{Northern~Sami}{NSM}\newfontlanguage{Northern~Tai}{NTA}
1657 \newfontlanguage{Esperanto}{NTO}\newfontlanguage{Nynorsk}{NYN}
1658 \newfontlanguage{Oji-Cree}{OCR}\newfontlanguage{Ojibway}{OJB}
1659 \newfontlanguage{Oriya}{ORI}\newfontlanguage{Oromo}{ORO}

1660 \newfontlanguage{Ossetian}{OSS}\newfontlanguage{Palestinian~Aramaic}{PAA}
1661 \newfontlanguage{Pali}{PAL}\newfontlanguage{Punjabi}{PAN}
1662 \newfontlanguage{Palpa}{PAP}\newfontlanguage{Pashto}{PAS}
1663 \newfontlanguage{Polytonic~Greek}{PGR}\newfontlanguage{Pilipino}{PIL}
1664 \newfontlanguage{Palaung}{PLG}\newfontlanguage{Polish}{PLK}
1665 \newfontlanguage{Provencal}{PRO}\newfontlanguage{Portuguese}{PTG}
1666 \newfontlanguage{Chin}{QIN}\newfontlanguage{Rajasthani}{RAJ}
1667 \newfontlanguage{R-Cree}{RCR}\newfontlanguage{Russian~Buriat}{RBU}
1668 \newfontlanguage{Riang}{RIA}\newfontlanguage{Rhaeto-Romanic}{RMS}
1669 \newfontlanguage{Romanian}{ROM}\newfontlanguage{Romany}{ROY}
1670 \newfontlanguage{Rusyn}{RSY}\newfontlanguage{Ruanda}{RUA}
1671 \newfontlanguage{Russian}{RUS}\newfontlanguage{Sadri}{SAD}
1672 \newfontlanguage{Sanskrit}{SAN}\newfontlanguage{Santali}{SAT}
1673 \newfontlanguage{Sayisi}{SAY}\newfontlanguage{Sekota}{SEK}
1674 \newfontlanguage{Selkup}{SEL}\newfontlanguage{Sango}{SGO}
1675 \newfontlanguage{Shan}{SHN}\newfontlanguage{Sibe}{SIB}
1676 \newfontlanguage{Sidamo}{SID}\newfontlanguage{Silte~Gurage}{SIG}
1677 \newfontlanguage{Skolt~Sami}{SKS}\newfontlanguage{Slovak}{SKY}
1678 \newfontlanguage{Slavey}{SLA}\newfontlanguage{Slovenian}{SLV}
1679 \newfontlanguage{Somali}{SML}\newfontlanguage{Samoan}{SMO}
1680 \newfontlanguage{Sena}{SNA}\newfontlanguage{Sindhi}{SND}
1681 \newfontlanguage{Sinhalese}{SNH}\newfontlanguage{Soninke}{SNK}
1682 \newfontlanguage{Sodo~Gurage}{SOG}\newfontlanguage{Sotho}{SOT}
1683 \newfontlanguage{Albanian}{SQI}\newfontlanguage{Serbian}{SRB}
1684 \newfontlanguage{Saraiki}{SRK}\newfontlanguage{Serer}{SRR}
1685 \newfontlanguage{South~Slavey}{SSL}\newfontlanguage{Southern~Sami}{SSM}
1686 \newfontlanguage{Suri}{SUR}\newfontlanguage{Svan}{SVA}
1687 \newfontlanguage{Swedish}{SVE}\newfontlanguage{Swadaya~Aramaic}{SWA}
1688 \newfontlanguage{Swahili}{SWK}\newfontlanguage{Swazi}{SWZ}
1689 \newfontlanguage{Sutu}{SXT}\newfontlanguage{Syriac}{SYR}
1690 \newfontlanguage{Tabasaran}{TAB}\newfontlanguage{Tajiki}{TAJ}
1691 \newfontlanguage{Tamil}{TAM}\newfontlanguage{Tatar}{TAT}
1692 \newfontlanguage{TH-Cree}{TCR}\newfontlanguage{Telugu}{TEL}
1693 \newfontlanguage{Tongan}{TGN}\newfontlanguage{Tigre}{TGR}
1694 \newfontlanguage{Tigrinya}{TGY}\newfontlanguage{Thai}{THA}
1695 \newfontlanguage{Tahitian}{THT}\newfontlanguage{Tibetan}{TIB}
1696 \newfontlanguage{Turkmen}{TKM}\newfontlanguage{Temne}{TMN}
1697 \newfontlanguage{Tswana}{TNA}\newfontlanguage{Tundra~Nenets}{TNE}
1698 \newfontlanguage{Tonga}{TNG}\newfontlanguage{Todo}{TOD}
1699 \newfontlanguage{Tsonga}{TSG}\newfontlanguage{Turoyo~Aramaic}{TUA}
1700 \newfontlanguage{Tulu}{TUL}\newfontlanguage{Tuvini}{TUV}
1701 \newfontlanguage{Twi}{TWI}\newfontlanguage{Udmurt}{UDM}
1702 \newfontlanguage{Ukrainian}{UKR}\newfontlanguage{Urdu}{URD}
1703 \newfontlanguage{Upper~Sorbian}{USB}\newfontlanguage{Uyghur}{UYG}
1704 \newfontlanguage{Uzbek}{UZB}\newfontlanguage{Venda}{VEN}
1705 \newfontlanguage{Vietnamese}{VIT}\newfontlanguage{Wa}{WA}
1706 \newfontlanguage{Wagdi}{WAG}\newfontlanguage{West-Cree}{WCR}
1707 \newfontlanguage{Welsh}{WEL}\newfontlanguage{Wolof}{WLF}
1708 \newfontlanguage{Tai~Lue}{XBD}\newfontlanguage{Xhosa}{XHS}
1709 \newfontlanguage{Yakut}{YAK}\newfontlanguage{Yoruba}{YBA}
1710 \newfontlanguage{Y-Cree}{YCR}\newfontlanguage{Yi~Classic}{YIC}

```

1711 \newfontlanguage{Yi~Modern}{YIM}\newfontlanguage{Chinese~Hong~Kong}{ZHH}
1712 \newfontlanguage{Chinese~Phonetic}{ZHP}\newfontlanguage{Chinese~Simplified}{ZHS}
1713 \newfontlanguage{Chinese~Traditional}{ZHT}\newfontlanguage{Zande}{ZND}
1714 \newfontlanguage{Zulu}{ZUL}

```

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

```

1715 \define@key[zf@feat]{Lang}{Turkish}[] {
1716   \fontspec_check_lang:nTF {TRK} {
1717     \c@zf@language\l_fontspec_strnum_int\relax
1718     \zf@update@family{+lang=Turkish}
1719     \tl_set:Nn \l_fontspec_lang_tl {TRK}
1720   } {
1721     \fontspec_check_lang:nTF {TUR} {
1722       \c@zf@language\l_fontspec_strnum_int\relax
1723       \zf@update@family{+lang=Turkish}
1724       \tl_set:Nn \l_fontspec_lang_tl {TUR}
1725     } {
1726       \fontspec_warning:nx {language-not-exist} {#1}
1727     }
1728   }
1729 }

```

Default

```

1730 \define@key[zf@feat]{Lang}{Default}[] {
1731   \zf@update@family{+lang=dflt}
1732   \tl_set:Nn \l_fontspec_lang_tl {DFLT}
1733   \c@zf@language=0\relax
1734 }

```

22.9.23 Raw feature string

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

```

1735 \define@key[zf]{options}{RawFeature}{
1736   \zf@update@family{+Raw:#1}
1737   \zf@update@ff{#1}
1738 }

```

22.10 Italic small caps

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

\sishape First, the commands for actually selecting italic small caps are defined. I use si
\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.

```

1739 \providecommand*\sidefault}{si}
1740 \DeclareRobustCommand{\sishape}{
1741   \not@math@alphabet\sishape\relax
1742   \fontshape\sidefault\selectfont
1743 }
1744 \DeclareTextFontCommand{\textsi}{\sishape}

```

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

```

1745 \newcommand*\zf@merge@shape}[3]{
1746   \edef\@tempa{#1}
1747   \edef\@tempb{#2}
1748   \ifx\f@shape\@tempb
1749     \ifcsname\f@encoding/\f@family/\f@series/#3\endcsname
1750       \edef\@tempa{#3}
1751   \fi
1752   \fi
1753   \fontshape{\@tempa}\selectfont
1754 }

```

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

```

\scshape
\upshape
1755 \DeclareRobustCommand \itshape {
1756   \not@math@alphabet\itshape\mathit
1757   \zf@merge@shape\itdefault\scdefault\sidefault
1758 }
1759 \DeclareRobustCommand \slshape {
1760   \not@math@alphabet\slshape\relax
1761   \zf@merge@shape\sldefault\scdefault\sidefault
1762 }
1763 \DeclareRobustCommand \scshape {
1764   \not@math@alphabet\scshape\relax
1765   \zf@merge@shape\scdefault\itdefault\sidefault
1766 }
1767 \DeclareRobustCommand \upshape {
1768   \not@math@alphabet\upshape\relax
1769   \zf@merge@shape\updefault\sidefault\scdefault
1770 }

```

22.11 Selecting maths fonts

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

`\zf@math` Everything here is performed `\AtBeginDocument` in order to overwrite euler's attempt. This means fontspec must be loaded *after* 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.

```

1771 \ifpackageloaded{euler}{\zf@package@euler@loadedtrue}
1772                        {\zf@package@euler@loadedfalse}
1773 \def\zf@math{
1774   \let\zf@font@warning\@font@warning
1775   \let\@font@warning\@font@info
1776   \ifpackageloaded{euler}{
1777     \ifzf@package@euler@loaded
1778       \zf@math@eulertrue
1779     \else
1780       \fontspec_error:n {euler-too-late}
1781     \fi
1782   }{}
1783   \ifpackageloaded{lucbmath}{\zf@math@lucidatrue}{}
1784   \ifpackageloaded{lucidabr}{\zf@math@lucidatrue}{}
1785   \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 re-define all of the maths glyphs in L^AT_EX's operators maths font to still go back to the legacy `cmr` font for all these random glyphs, unless a separate maths font package has been loaded instead.

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

```

1786 \DeclareSymbolFont{legacymaths}{OT1}{cmr}{m}{n}
1787 \SetSymbolFont{legacymaths}{bold}{OT1}{cmr}{bx}{n}
1788 \DeclareMathAccent{\acute}{\mathalpha}{legacymaths}{19}
1789 \DeclareMathAccent{\grave}{\mathalpha}{legacymaths}{18}
1790 \DeclareMathAccent{\ddot}{\mathalpha}{legacymaths}{127}
1791 \DeclareMathAccent{\tilde}{\mathalpha}{legacymaths}{126}
1792 \DeclareMathAccent{\bar}{\mathalpha}{legacymaths}{22}
1793 \DeclareMathAccent{\breve}{\mathalpha}{legacymaths}{21}
1794 \DeclareMathAccent{\check}{\mathalpha}{legacymaths}{20}
1795 \DeclareMathAccent{\hat}{\mathalpha}{legacymaths}{94} % too bad, euler
1796 \DeclareMathAccent{\dot}{\mathalpha}{legacymaths}{95}
1797 \DeclareMathAccent{\mathring}{\mathalpha}{legacymaths}{23}

```

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

```

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

% amsmath.sty:
\renewcommand{\colon}{\nobreak\mskip2mu\mathpunct{}\nonscript

```



```
\mkern-\thinmuskip{:}\mskip6mu plus 1mu\relax}
```

```
% euler.sty:
```

```
\DeclareMathSymbol{:}\mathrel {EulerFraktur}{\textfont"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.

```
1798 \begingroup
```

```
1799 \mathchardef\@tempa="603A \relax
```

```
1800 \let\next\egroup
```

```
1801 \ifx\colon\@tempa
```

```
1802 \DeclareMathSymbol{\colon}\mathpunct{legacymaths}{58}
```

```
1803 \fi
```

```
1804 \endgroup
```

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

```
1805 \ifzf@math@euler\else
```

```
1806 \DeclareMathSymbol{!}\mathclose{legacymaths}{33}
```

```
1807 \DeclareMathSymbol{:}\mathrel {legacymaths}{58}
```

```
1808 \DeclareMathSymbol{;}\mathpunct{legacymaths}{59}
```

```
1809 \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.

```
1810 \ifzf@math@lucida\else
```

```
1811 \DeclareMathSymbol{0}\mathalpha{legacymaths}{‘0}
```

```
1812 \DeclareMathSymbol{1}\mathalpha{legacymaths}{‘1}
```

```
1813 \DeclareMathSymbol{2}\mathalpha{legacymaths}{‘2}
```

```
1814 \DeclareMathSymbol{3}\mathalpha{legacymaths}{‘3}
```

```
1815 \DeclareMathSymbol{4}\mathalpha{legacymaths}{‘4}
```

```
1816 \DeclareMathSymbol{5}\mathalpha{legacymaths}{‘5}
```

```
1817 \DeclareMathSymbol{6}\mathalpha{legacymaths}{‘6}
```

```
1818 \DeclareMathSymbol{7}\mathalpha{legacymaths}{‘7}
```

```
1819 \DeclareMathSymbol{8}\mathalpha{legacymaths}{‘8}
```

```
1820 \DeclareMathSymbol{9}\mathalpha{legacymaths}{‘9}
```

```
1821 \DeclareMathSymbol{\Gamma}\mathalpha{legacymaths}{0}
```

```
1822 \DeclareMathSymbol{\Delta}\mathalpha{legacymaths}{1}
```

```
1823 \DeclareMathSymbol{\Theta}\mathalpha{legacymaths}{2}
```

```
1824 \DeclareMathSymbol{\Lambda}\mathalpha{legacymaths}{3}
```

```
1825 \DeclareMathSymbol{\Xi}\mathalpha{legacymaths}{4}
```

```
1826 \DeclareMathSymbol{\Pi}\mathalpha{legacymaths}{5}
```

```
1827 \DeclareMathSymbol{\Sigma}\mathalpha{legacymaths}{6}
```

```

1828 \DeclareMathSymbol{\Upsilon}{\mathalpha}{legacymaths}{7}
1829 \DeclareMathSymbol{\Phi}{\mathalpha}{legacymaths}{8}
1830 \DeclareMathSymbol{\Psi}{\mathalpha}{legacymaths}{9}
1831 \DeclareMathSymbol{\Omega}{\mathalpha}{legacymaths}{10}
1832 \DeclareMathSymbol{+}{\mathbin}{legacymaths}{43}
1833 \DeclareMathSymbol{=}{\mathrel}{legacymaths}{61}
1834 \DeclareMathDelimiter{()}{\mathopen}{legacymaths}{40}{largesymbols}{0}
1835 \DeclareMathDelimiter{)}{\mathclose}{legacymaths}{41}{largesymbols}{1}
1836 \DeclareMathDelimiter{[]}{\mathopen}{legacymaths}{91}{largesymbols}{2}
1837 \DeclareMathDelimiter{[]}{\mathclose}{legacymaths}{93}{largesymbols}{3}
1838 \DeclareMathDelimiter{/}{\mathord}{legacymaths}{47}{largesymbols}{14}
1839 \DeclareMathSymbol{\mathdollar}{\mathord}{legacymaths}{36}
1840 \fi
1841 \fi

```

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

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

```

1842 \DeclareSymbolFont{operators}\zf@enc\zf@rmmaths\mddefault\updefault
1843 \SetSymbolFont{operators}{normal}\zf@enc\zf@rmmaths\mddefault\updefault
1844 \SetMathAlphabet\mathrm{normal}\zf@enc\zf@rmmaths\mddefault\updefault
1845 \SetMathAlphabet\mathit{normal}\zf@enc\zf@rmmaths\mddefault\itdefault
1846 \SetMathAlphabet\mathbf{normal}\zf@enc\zf@rmmaths\bfdefault\updefault
1847 \SetMathAlphabet\mathsf{normal}\zf@enc\zf@sfmtmaths\mddefault\updefault
1848 \SetMathAlphabet\mathtt{normal}\zf@enc\zf@ttmaths\mddefault\updefault
1849 \SetSymbolFont{operators}{bold}\zf@enc\zf@rmmaths\bfdefault\updefault
1850 \ifdefined\zf@rmboldmaths
1851 \SetMathAlphabet\mathrm{bold}\zf@enc\zf@rmboldmaths\mddefault\updefault
1852 \SetMathAlphabet\mathbf{bold}\zf@enc\zf@rmboldmaths\bfdefault\updefault
1853 \SetMathAlphabet\mathit{bold}\zf@enc\zf@rmboldmaths\mddefault\itdefault
1854 \else
1855 \SetMathAlphabet\mathrm{bold}\zf@enc\zf@rmmaths\bfdefault\updefault
1856 \SetMathAlphabet\mathit{bold}\zf@enc\zf@rmmaths\bfdefault\itdefault
1857 \fi
1858 \SetMathAlphabet\mathsf{bold}\zf@enc\zf@sfmtmaths\bfdefault\updefault
1859 \SetMathAlphabet\mathtt{bold}\zf@enc\zf@ttmaths\bfdefault\updefault
1860 \let\font@warning\zf@font@warning}

```

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

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

```

1861 \def\zf@math@maybe{
1862 \ifpackageloaded{anttor}{
1863 \ifx\define@antt@mathversions a\zf@mathfalse\fi}{
1864 \ifpackageloaded{arev}{\zf@mathfalse}{

```

```

1865 \ifpackageloaded{eulervm}{\@zf@mathfalse{}}
1866 \ifpackageloaded{mathdesign}{\@zf@mathfalse{}}
1867 \ifpackageloaded{concmath}{\@zf@mathfalse{}}
1868 \ifpackageloaded{cmbright}{\@zf@mathfalse{}}
1869 \ifpackageloaded{mathesf}{\@zf@mathfalse{}}
1870 \ifpackageloaded{gfsartemisia}{\@zf@mathfalse{}}
1871 \ifpackageloaded{gfsneoellenic}{\@zf@mathfalse{}}
1872 \ifpackageloaded{iwona}{
1873   \ifx\define@iwona@mathversions a\@zf@mathfalse\fi{}}
1874 \ifpackageloaded{kpfonts}{\@zf@mathfalse{}}
1875 \ifpackageloaded{kmath}{\@zf@mathfalse{}}
1876 \ifpackageloaded{kurier}{
1877   \ifx\define@kurier@mathversions a\@zf@mathfalse\fi{}}
1878 \ifpackageloaded{fouriernc}{\@zf@mathfalse{}}
1879 \ifpackageloaded{fourier}{\@zf@mathfalse{}}
1880 \ifpackageloaded{mathpazo}{\@zf@mathfalse{}}
1881 \ifpackageloaded{mathptmx}{\@zf@mathfalse{}}
1882 \ifpackageloaded{MinionPro}{\@zf@mathfalse{}}
1883 \ifpackageloaded{unicode-math}{\@zf@mathfalse{}}
1884 \ifpackageloaded{breqn}{\@zf@mathfalse{}}
1885 \if@zf@math
1886   \fontspec_info:n {setup-math}
1887   \zf@math
1888   \fi
1889 }
1890 \AtBeginDocument{\zf@math@maybe}

```

22.12 Finishing up

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

```

1891 \if@zf@configfile
1892   \InputIfFileExists{fontspec.cfg}
1893   {\typeout{fontspec.cfg~ loaded.}}
1894   {\typeout{No~ fontspec.cfg~ file~ found;~ no~ configuration~ loaded.}}
1895 \fi

```

The end! Thanks for coming.

Part VII

fontspec.lua

First we define some metadata.

```
1 fontspec      = { }
2
3 fontspec.module = {
4   name        = "fontspec",
5   version     = 2.0,
6   date        = "2009/12/04",
7   description  = "Advanced font selection for LuaLaTeX.",
8   author      = "Khaled Hosny",
9   copyright   = "Khaled Hosny",
10  license     = "LPPL"
11 }
12
13 luatexbase.provides_module(fontspec.module)
14
```

Some utility functions

```
15
16 utf = unicode.utf8
17
18 function fontspec.log (...) luatexbase.module_log (fontspec.module.name, string.format(...))
19 function fontspec.warning(...) luatexbase.module_warning(fontspec.module.name, string.format(...))
20 function fontspec.error (...) luatexbase.module_error (fontspec.module.name, string.format(...))
21
22 function fontspec.sprint (...) tex.sprint(luatexbase.catcodetables['latex-package'], ...) end
23
```

The following functions check for existence of certain script, language or feature in a given font.

```
24
25 local function check_script(id, script)
26   local s = string.lower(script)
27   if id and id ~= 0 then
28     local otfddata = fonts.ids[id].shared.otfddata
29     if otfddata then
30       local features = otfddata.luaTeX.features
31       for i,_ in pairs(features) do
32         for j,_ in pairs(features[i]) do
33           if features[i][j][s] then
34             fontspec.log("script '%s' exists in font '%s'",
35                           script, fonts.ids[id].fullname)
36             return true
37           end
38         end
39       end
40     end
41   end
end
```

```

42 end
43
44 local function check_language(id, language, script)
45     local s = string.lower(script)
46     local l = string.lower(language)
47     if id and id ~= 0 then
48         local otfddata = fonts.ids[id].shared.otfddata
49         if otfddata then
50             local features = otfddata.luatex.features
51             for i,_ in pairs(features) do
52                 for j,_ in pairs(features[i]) do
53                     if features[i][j][s] and features[i][j][s][l] then
54                         fontspec.log("language '%s' for script '%s' exists in font '%s'",
55                                     language, script, fonts.ids[id].fullname)
56                         return true
57                     end
58                 end
59             end
60         end
61     end
62 end
63
64 local function check_feature(id, feature, language, script)
65     local s = string.lower(script)
66     local l = string.lower(language)
67     local f = string.lower(feature:gsub("[+-]", ""))
68     if id and id ~= 0 then
69         local otfddata = fonts.ids[id].shared.otfddata
70         if otfddata then
71             local features = otfddata.luatex.features
72             for i,_ in pairs(features) do
73                 if features[i][f] and features[i][f][s] then
74                     if features[i][f][s][l] == true then
75                         fontspec.log("feature '%s' for language '%s' and script '%s' exists in font '%s'",
76                                     feature, language, script, fonts.ids[id].fullname)
77                         return true
78                     end
79                 end
80             end
81         end
82     end
83 end
84

```

This function takes a font csname (without the leading slash) and returns its internal font id. Since LuaTeX 0.47, there is a built in `font.id()` function.

```

85
86 local function font_id(str)
87     local id
88     if tex.luatexversion == 47 then
89         id = font.id(str)
90     else

```

```

91         id = token.create(str)[2]
92     end
93     return id
94 end
95

```

The following are the function that get called from T_EX end.

```

96
97 local function tempswattrue() fontspec.sprint([[ \@tempswattrue]]) end
98 local function tempswafalse() fontspec.sprint([[ \@tempswafalse]]) end
99
100 function fontspec.check_ot_script(fnt, script)
101     if check_script(font_id(fnt), script) then
102         tempswattrue()
103     else
104         tempswafalse()
105     end
106 end
107
108 function fontspec.check_ot_lang(fnt, lang, script)
109     if check_language(font_id(fnt), lang, script) then
110         tempswattrue()
111     else
112         tempswafalse()
113     end
114 end
115
116 function fontspec.check_ot_feat(fnt, feat, lang, script)
117     for _, f in ipairs { "+trep", "+tlig", "+anum" } do
118         if feat == f then
119             tempswattrue()
120             return
121         end
122     end
123     if check_feature(font_id(fnt), feat, lang, script) then
124         tempswattrue()
125     else
126         tempswafalse()
127     end
128 end
129

```

Get font dimens for \fontspec_calc_scale:n (despite the name of this function, it doesn't return any arbitrary font dimen, only \dimen5 and \dimen8.)

```

130
131 function fontspec.get_dimen(fontdimen, csname)
132     local id, h, em, pt
133     if csname == "font" then
134         id = font.current()
135     else
136         id = font_id(csname)
137     end

```

```

138     if fontdimen == 8 then
139         h = fonts.ids[id].shared.otfdata.pfminfo.os2_capheight
140     elseif fontdimen == 5 then
141         h = fonts.ids[id].shared.otfdata.pfminfo.os2_xheight
142     end
143     em = fonts.ids[id].shared.otfdata.metadata.units_per_em
144     pt = fonts.ids[id].size / 65536
145     tex.sprint(string.format("%spt", (h/em)*pt))
146 end
147
148 function fontspec.charglyph(char, csname)
149     local id, c
150     if char then
151         if utf.len(char) == 1 then
152             c = utf.byte(utf.char(char:gsub("'", "'0x')))
153         else
154             c = utf.byte(char)
155         end
156     end
157     if csname then
158         id = font_id(csname)
159     else
160         id = font.current()
161     end
162
163     if font.fonts[id]["characters"][c] then
164         return font.fonts[id]["characters"][c].index
165     else
166         return 0
167     end
168 else
169     return 0
170 end
171 end
172

```

Part VIII

fontspec-patches.sty

```
1 \ExplSyntaxOn
```

22.13 Unicode footnote symbols

```
2 \RequirePackage{fixltx2e}[2006/03/24]
```

22.14 Emph

```
\em   Redefinition of {\em ...} and \emph{...} to use NFSS info to detect when the inner
\emph shape should be used.
\emshape 3 \DeclareRobustCommand \em {
\emminnershape 4 \@nomath\em
5 \tl_if_eq:xxTF \f@shape \itdefault \emminnershape \emshape
6 }
7 \DeclareTextFontCommand{\emph}{\em}
8 \let\emshape\itshape
9 \let\emminnershape\upshape
```

22.15 \-

\- This macro is courtesy of Frank Mittelbach and the L^AT_EX 2_ε source code.

```
10 \DeclareRobustCommand{\-}{%
11 \discretionary{%
12 \char\ifnum\hyphenchar\font\z@
13 \xlx@defaultthyphenchar
14 \else
15 \hyphenchar\font
16 \fi}{}}{}
17 \def\xlx@defaultthyphenchar{'\-}
```

22.16 Verbatims

Many thanks to Apostolos Syropoulos for discovering this problem and writing the redefinition of L^AT_EX's verbatim environment and \verb* command.

\xxt@visiblespace Print U+2434: OPEN BOX, which is used to visibly display a space character.

```
18 \def\xxt@visiblespace{
19 \iffontchar\font"2423
20 \expandafter\textvisiblespace
21 \else
22 \expandafter\xxt@visiblespace@fallback
23 \fi
24 }
```

\xxt@visiblespace@fallback If the current font doesn't have u2434, use Latin Modern Mono instead.

```
25 \def\xxt@visiblespace@fallback{
26 {
```



```

27 \usefont{EU1}{lmtt}{\f@series}{\f@shape}
28 \textvisiblespace
29 }
30 }

```

`\xxt@vprintspaces` Helper macro to turn spaces active and print visible space instead.

```

31 \begingroup
32 \catcode'\~=\active
33 \gdef\xxt@vprintspaces{\catcode'\~\active\let \xxt@visiblespace}
34 \endgroup

```

`\verb` Redefine `\verb` to use `\xxt@vprintspaces`.

```

\verb* 35 \def\verb{
36 \relax\ifmmode\hbox\else\leavevmode\null\fi
37 \bgroup
38 \verb@eol@error \let\do\@makeother \dospecials
39 \verbatim@font\@noligs
40 \@ifstar\@sverb\@verb
41 }
42 \def\@sverb{\xxt@vprintspaces@sverb}

```

It's better to put small things into `\AtBeginDocument`, so here we go:

```

43 \AtBeginDocument{
44 \fontspec_patch_verbatim:
45 \fontspec_patch_moreverb:
46 \fontspec_patch_fancyvrb:
47 \fontspec_patch_listings:
48 }

```

`verbatim*` With the `verbatim` package.

```

49 \cs_set:Npn \fontspec_patch_verbatim: {
50 \@ifpackageloaded{verbatim}{
51 \@namedef{verbatim*}{
52 \begingroup\@verbatim\xxt@vprintspaces\verbatim@start
53 }
54 }{

```

This is for vanilla LaTeX.

```

55 \@namedef{verbatim*}{\@verbatim\xxt@vprintspaces\@sxverbatim}
56 }
57 }

```

`listingcont*` This is for `moreverb`. The main `listing*` environment inherits this definition.

```

58 \cs_set:Npn \fontspec_patch_moreverb: {
59 \@ifpackageloaded{moreverb}{
60 \@namedef{listingcont*}{
61 \def\verbatim@processline{
62 \thelisting@line \global\advance\listing@line\c_one
63 \the\verbatim@line\par
64 }
65 \@verbatim\xxt@vprintspaces\verbatim@start
66 }
67 }{}}

```

listings and fancvrb make things nice and easy:

```
68 \cs_set:Npn \fontspec_patch_fancyvrb: {  
69   \@ifpackageloaded{fancyvrb}{  
70     \let\FancyVerbSpace\xxt@visibleSPACE  
71   }{}  
72 }  
  
73 \cs_set:Npn \fontspec_patch_listings: {  
74   \@ifpackageloaded{listings}{  
75     \let\lst@visibleSPACE\xxt@visibleSPACE  
76   }{}  
77 }
```

Part IX

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 % Nothing here!
6 % I have absolutely no idea whether backwards compatibility,
7 % of the sort that was previously populated here, is important
8 % for version 2.
9
```

Part X

Example documents

```
1 \luatex\% !TEX TS-program = LuaLaTeX
2 \xetex\% !TEX TS-program = XeLaTeX
3
4 \documentclass{article}
5
6 \luatex\usepackage{fontspec}
7 \xetex\usepackage{xltextra}
8
9 \setmainfont[Ligatures=TeX]{TeX Gyre Pagella}
10 \setsansfont[Ligatures=TeX,Scale=MatchLowercase]{TeX Gyre Heros}
11 \setmonofont[Scale=MatchLowercase]{Inconsolata}
12
13 \frenchspacing % TeX's default is a little old-fashioned...
14
15 \begin{document}
16 \pagestyle{empty}
17
18 \section*{The basics of the \textsf{fontspec} package}
19
20 The \textsf{fontspec} package enables automatic font selection
21 for \LaTeX{} documents typeset with Xe\TeX{} or \LuaTeX.
22 The basic command is
23
24 {\centering \verb|\fontspec[font features]{font display name}|.\par}
25
26 The default, sans serif, and typewriter fonts may be set with the
27 \verb|\setmainfont|, \verb|\setsansfont| and \verb|\setmonofont|
28 commands, respectively, as shown in the preamble. They take the
29 same syntax as the \verb|\fontspec| package. All expected font
30 shapes are available:
31
32 \begin{center}
33   {\itshape Italics and \scshape small caps\dots}\
34   {\sffamily\bfseries Bold sans serif and \itshape bold italic sans serif\dots}
35 \end{center}
36
37 Text fonts in maths mode are also changed (e.g., notice the cosine function in
38 '$\cos(n\pi)=\pm 1$') but only if the roman and sans serif fonts are set in
39 the preamble; \verb|\setmainfont| will not affect these maths mode fonts when
40 called mid-document.
41 Maths symbols themselves are not affected.
42
43 Notice the font features used to load the default fonts in the preamble.
44 The first, \verb|Ligatures=TeX|, enables regular \TeX{} ligatures like
45 \verb|'---'| for '---'.
46 The second, \verb|Scale=MatchLowercase|, automatically scales the fonts to
47 the same x-height.
```

48
49 Please see the complete `\textsf{fontspec}` documentation for further
50 information.
51
52 `\end{document}`

Change History

| | |
|---|----|
| v1.0 | |
| General: Initial version. | 29 |
| v1.1 | |
| General: Name change to fontspec. | 29 |
| \setmainfont: Implemented (with friends). | 32 |
| v1.10 | |
| General: Color brought back into the .sty | 51 |
| New feature LetterSpace. | 50 |
| Some babel encoding problems resolved. | 31 |
| \addfontfeatures: Saved family information macro changes. | 33 |
| \zf@fontspec: Saved family info split into two (now three) macros. | 38 |
| Space zapped from L ^A T _E X family name due to various problems. | 38 |
| \zf@make@feature: Removed embarrassing space after warnings. | 43 |
| \zf@math: Added lucimatx checking. (Not really tested, though.) | 61 |
| Fixed Lucida bug (missing \else) | 61 |
| v1.11 | |
| General: HyphenChar checks its input now. | 50 |
| Added better support for Turkish language selection. | 60 |
| OpenType Variant fixed. | 54 |
| \emph: Redefinitions moved to xltextra. | 61 |
| \newfontface: Name change from \newfontfamily. | 33 |
| \newfontlanguage: Fixed \c@zf@language setting not being global. | 35 |
| \newfontscript: Fixed \c@zf@script setting not being global. | 35 |
| \zf@fontspec: Ensure bold/italic fonts are loaded with the same renderer as the regular font even if unspecified. | 38 |
| \zf@wordspace@parse: Improved saving \fontdimen stuff to \zd@adjust(also see PunctuationSpace). | 50 |
| v1.12 | |
| General: BoldFont, etc., flags \zf@nobf conditional false rather than assuming it implicitly. This allows, <i>e.g.</i> , empty BoldFont to be overloaded. | 47 |
| Finally, use the EU1 font encoding. | 31 |
| New feature ExternalLocation for loading external fonts. | 46 |
| Package option for disabling the EU1 encoding. | 30 |
| \addfontfeatures: Now use grouping to restore \zf@default@option change. | 33 |
| \zf@make@aat@feature@string: Fixed result of \XeTeXfeaturename output change (empty string if odd non-exclusive selector). | 44 |
| Removed \@thisfontfeature macro; replaced with \@tempa. | 44 |
| v1.13 | |
| General: RawFeature added | 60 |
| SizeFeatures: Beginnings of support for different font features for different font sizes, required by unicode-math. | 48 |
| \zf@DeclareFontShape: New feature SizeFeatures implemented. | 40 |
| \zf@make@feature: Warns (and fixes bug) when an AAT feature is requested for an ICU font and vice versa. | 43 |
| \zf@partial@fontname: Folded in the \@tempa \let command, changing the syntax. | 48 |
| v1.15 | |
| General: RotatedGlyphs now works for ICU fonts. | 55 |
| v1.16 | |
| \zf@math@maybe: Maths setup warning sent to the log file instead. | 64 |

| | |
|--|----|
| v1.17 | |
| General: [quiet] now suppresses warnings. | 30 |
| New features FakeSlant, FakeStretch, FakeBold. | 51 |
| New package option [silent]. | 30 |
| \zf@DeclareFontShape: slshape substitution bug fix (thanks Ulrike). | 40 |
| slshape substitution is now silent. | 40 |
| \zf@fontspec: New feature: UprightFont. | 38 |
| v1.2 | |
| General: Initial OpenType support. | 29 |
| Support for Scale. | 49 |
| v1.3 | |
| General: More OpenType support. | 29 |
| Support for Mapping and Colour. | 51 |
| \defaultfontfeatures: Implemented. | 33 |
| \newAATfeature: Implemented. | 34 |
| \newfontfeature: Implemented. | 34 |
| v1.3a | |
| General: Bug fix for OpenType small caps. | 52 |
| v1.4 | |
| General: Support for Weight and Width AAT features. | 51 |
| \defaultfontfeatures: Name changed from \setdefaultoptions. | 33 |
| \zf@math: Selects the default \mathXX fonts. | 61 |
| v1.5 | |
| General: New options for arbitrary bold/italic shapes. | 47 |
| \addfontfeatures: Implemented. | 33 |
| \zf@fontspec: Added code for choosing arbitrary bold/italic fonts. | 38 |
| Checks if the font family has already been defined. | 38 |
| NFSS specifiers now take the default values. | 38 |
| \zf@make@font@shapes: Absorbed font-checking from \zf@fontspec. | 39 |
| v1.5a | |
| \zf@math: Added fix for Computer Modern maths. | 61 |
| v1.6 | |
| General: Bold option aliased to BoldFont. | 47 |
| LetterCase is now Letters and options changed appropriately. | 52 |
| Scale feature now updates family name. | 49 |
| All AAT Fractions features offered. | 54 |
| New OpenType feature: Language | 56 |
| New OpenType feature: Script | 55 |
| OpenType letters features: PetiteCaps and PETITECAPS. | 52 |
| OpenType ligature features: Contextual and Historical. | 52 |
| OpenType stylistic sets supports under the Variant option. | 54 |
| \addfontfeatures: Removed \relaxing of temporary macros. | 33 |
| \fontspec: Removed \zf@currfont (unnecessary) | 31 |
| \newfontface: Implemented. | 33 |
| \newfontfeature: newff counter now uses LaTeX methods rather than primitive TeX. I don't know if there is any advantage to this. | 34 |
| \setmainfont: Changed \rmdefault, etc., assigning to use \let directly. | 32 |
| \zf@fontspec: Added code for choosing arbitrary bold/italic font features. | 38 |
| Writes some info to the .log file | 38 |
| \zf@get@feature@requests: Removed the space between the comma and \zf@options when it's concatenated with the defaults. | 41 |

| | |
|--|----|
| \zf@math: Removed mathtime support since XeTeX doesn't handle virtual fonts. Why did I put it in in the first place? | 61 |
| v1.7 | |
| General: Style feature renamed from StyleOptions. | 54 |
| AAT Numbers:SlashedZero. | 53 |
| New feature: Annotation | 55 |
| New feature: CharacterShape | 55 |
| New feature: CharacterWidth | 55 |
| New feature: OpticalSize; works with both OpenType and MM fonts. | 51 |
| OpenType Alternate Fractions feature. | 54 |
| OpenType Alternate now only AAT. | 54 |
| Removed AAT check for weight/width axes (could also be Multiple Master) | 51 |
| \zf@define@feature@option: Implemented for the bulk of the feature processing code. | 43 |
| \zf@fontspec: Optional argument now mandatory. | 38 |
| \zf@make@aat@feature@string: Changed some \edefs to \let | 44 |
| Removed third argument; always saves the feature string in \zf@thisfontfeature | 44 |
| \zf@make@feature: Accommodation of the \zf@thisfontfeature change. | 43 |
| \zf@make@font@shapes: Changed some \edefs to \let. | 39 |
| Support for the OpticalSize feature. | 39 |
| \zf@make@smallcaps: Accommodation of the \zf@thisfontfeature change. | 42 |
| \zf@set@font@type: Added 'MM' font type; tests true, <i>e.g.</i> , with Skia & Minion MM. Used with the OpticalSize feature. | 38 |
| Removed exclusivity from font type (AAT, OpenType) check, since fonts can be both. | 38 |
| Removed various \count255s. | 38 |
| \zf@update@ff: Fix for featureless fonts (<i>e.g.</i> , the MS fonts) being ignored. | 42 |
| v1.8 | |
| \setmathrm: Implemented (with friends). | 32 |
| \zf@math: Added support for user-specified \mathrm and others. | 61 |
| Finally fixed legacy maths font issues. Also checks that euler.sty is loaded in the right order. | 61 |
| v1.8a | |
| \zf@math: Added conditional to \colon math symbol (incompatibility with lucida and amsmath) | 61 |
| v1.9 | |
| General: CharacterShape now CJKShape | 55 |
| SMALLCAPS option changed to UppercaseSmallCaps to facilitate option normalisation (to come). Similarly for PETITECAPS. | 52 |
| Swashes feature changed to Contextuals. Option of this feature Contextual changed to Swash, for obvious reasons. | 53 |
| TextSpacing now CharacterWidth, with associated option names' change. | 55 |
| Alternate/Variant options can be assigned names. | 54 |
| New Scale options: MatchLowercase and MatchUppercase. | 49 |
| New feature HyphenChar. | 50 |
| New feature Kerning. | 53 |
| New feature PunctuationSpace. | 50 |
| New feature UprightFeatures. | 47 |
| New feature Vertical. | 55 |
| New feature WordSpace. | 49 |
| New features SmallCapsFont and SmallCapsFeatures. | 47 |

| | |
|--|----|
| Package options (no)config, quiet implemented. | 65 |
| \addfontfeatures: Added \ignorespaces to make it invisible. | 33 |
| Changed \fontspec call to \@fontspec so that \ignorespaces isn't called unnecessarily. | 33 |
| \aliasfontfeature: Implemented. | 34 |
| \aliasfontfeatureoption: Implemented. | 34 |
| \emph: Redefined \em in order for nested emphases to work. | 61 |
| \fontspec: Added \ignorespaces to make it invisible. | 31 |
| \keyval@alias@key: Implemented. | 43 |
| \multi@alias@key: Implemented for \aliasfontfeature. | 43 |
| \newAATfeature: Replacement for \newfeaturecode. | 34 |
| \newfontlanguage: Implemented. | 35 |
| \newfontscript: Implemented. | 35 |
| \newICUfeature: Implemented. | 34 |
| \zf@calc@scale: Implemented for auto-scaling options. | 49 |
| \zf@check@ot@feat: Implemented. | 46 |
| \zf@check@ot@lang: Implemented. | 45 |
| \zf@check@ot@script: Implemented. | 45 |
| \zf@DeclareFontShape: Implemented as wrapper for \DeclareFontShape. | 40 |
| Slanted/italic shape substitution implemented. | 40 |
| \zf@fontspec: Absorbed the comma into \zf@. .options as to be more efficient when they are not defined. | 38 |
| Abstracted the long family name so the NFSS family is simple. | 38 |
| Incorporated \zf@get@feature@requests argument change. | 38 |
| Incorporated \zf@make@font@shapes change; removed \zf@options storage macro. | 38 |
| \zf@get@feature@requests: Absorbed comma into \zf@default@options, making \zf@current@options redundant. | 41 |
| Added an argument to eliminate the \zf@options macro. | 41 |
| Removed init stuff. | 41 |
| \zf@init: Taken from \zf@get@feature@requests. | 41 |
| \zf@make@feature: Now checks for OpenType feature. | 43 |
| \zf@make@font@shapes: \zf@scale@str eliminated. | 39 |
| Absorbed \IfEqFonts. | 39 |
| Added argument for \zf@get@feature@requests change. | 39 |
| Added code for SmallCaps... features. | 39 |
| Added logging of /B, /I, /BI failure. | 39 |
| Changed input syntax. | 39 |
| Incorporated \sidefault test into the \DeclareFontShape argument directly now that it's fully expanded. | 39 |
| Made local to hide \zf@fontname changes. | 39 |
| Removed \zf@scshape macro. | 39 |
| Removed \nfss@catcodes wrapper. | 39 |
| \zf@make@smallcaps: Now uses \zf@check@ot@feat. | 42 |
| \zf@math: Maths hex numbers converted to decimal. | 61 |
| Suppresses harmless maths font encoding warnings. | 64 |
| \zf@partial@fontname: Implemented. | 48 |
| \zf@update@family: Now fully expands arguments. | 41 |
| \zf@update@ff: Removed ridiculous \zf@feature@separator code. | 42 |
| \zf@v@strnum: Implemented. | 45 |
| \zf@wordspace@parse: Implemented. | 50 |

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 | |
|-------------------------------|---|
| \- | <u>10</u> , 782 |
| \@sverb | 40, 42 |
| \@empty | 267, 271, 352, 367, 531, 577, 588, 598, 608–610, 628, 629, 665, 677, 680, 748, 750, 789, 794, 837, 884, 894, 899, 902, 907, 913, 914, 1051, 1062, 1085, 1091, 1125, 1129, 1170, 1205, 1416, 1423 |
| \@firstofone | 708 |
| \@font@info | 1775 |
| \@font@warning | 1774, 1775 |
| \@ifpackageloaded | 50, 59, 69, 74, 1771, 1776, 1783–1785, 1862, 1864–1872, 1874–1876, 1878–1884 |
| \@ifstar | 40 |
| \@makeother | 38 |
| \@namedef | 51, 55, 60 |
| \@nameuse | 535 |
| \@ne | 524, 939, 959, 983 |
| \@noligs | 39 |
| \@nomath | 4 |
| \@onlypreamble | 248–251 |
| \@sverb | 42 |
| \@sxverbatim | 55 |
| \@tempa | 45, 367, 529, 531, 664, 665, 667, 671, 794, 795, 837, 841, 883, 884, 900, 902, 1050, 1051, 1061, 1062, 1084, 1085, 1169, 1170, 1197, 1199, 1746, 1750, 1753, 1799, 1801 |
| \@tempb | 670, 671, 886, 889, 891, 894, 895, 899, 900, 1198, 1199, 1205, 1243, 1747, 1748 |
| \@tempcntb | 932, 934, 937, 951, 954, 957, 973, 977, 981 |
| \@tempdima | 1148, 1150, 1171–1173, 1175, 1181, 1188, 1189 |
| \@tempdimb .. | 1149, 1150, 1172, 1176, 1182 |
| \@tempdimc .. | 1150, 1151, 1173, 1177, 1183 |
| \@tempfonta | 666, 667 |
| \@tempfontb | 669, 670, 675 |
| \@tempswafalse .. | 98, 605, 606, 933, 953, 976 |
| \@tempswatruetrue | 97, 604, 936, 956, 980 |
| \@verb | 40 |
| \@verbatim | 52, 55, 65 |
| \@zf@configfilefalse | 185 |
| \@zf@configfiletrue | 184 |
| \@zf@mathfalse | 183, 1863–1871, 1873–1875, 1877–1884 |
| \@zf@mathtrue | 182 |
| \@ | 33, 65, 81, 92–94, 100, 113, 118, 123, 128, 129, 138, 139, 156 |
| \~ | 32, 33 |
| A | |
| \active | 32, 33 |
| \acute | 1788 |
| \addfontfeature | 100, 284 |
| \addfontfeatures | <u>268</u> |
| \advance | 62, 523, 939, 959, 983 |
| \aliasfontfeature | <u>312</u> , 1005, 1428 |
| \aliasfontfeatureoption | <u>312</u> |
| \AtBeginDocument | 43, 1890 |
| B | |
| \bar | 1792 |
| \begin | 15, 32 |
| \begingroup ... | 31, 52, 270, 500, 663, 1798 |
| \bfdefault | 579, 582, 612, 615, 619, 623, 631, 635, 1846, 1849, 1852, 1855, 1856, 1858, 1859 |
| \bfseries | 34 |
| \bgroup | 37 |
| \bool_if:NF | 1008 |
| \bool_if:NTF | 568 |
| \bool_if:nTF | 918 |
| \bool_new:N | 997 |
| \bool_set_false:N | 564 |
| \bool_set_true:N | 566, 1001 |
| \breve | 1793 |
| C | |
| \c@zf@index | . 41, 933–935, 937, 939, 952, 954, 955, 957, 959, 975, 977, 979, 981, 983 |
| \c@zf@language | 43, 344, 399, 417, 546, 973, 978, 1717, 1722, 1733 |
| \c@zf@newff | 40, 287 |
| \c@zf@script | 42, 327, 397, 416, 445, 461, 544, 951, 955, 973, 978 |
| \c_one | 62 |

| | |
|-----------------------------------|--|
| environments: | |
| listingcont* | 58 |
| verbatim* | 49 |
| \ExecuteOptions | 194 |
| \exp_after:wN | 924, 925 |
| \exp_not:N . | 48, 52, 258, 259, 273, 719, 725 |
| \exp_not:V | 753 |
| \expandafter | 20, 22, 523, 526, 531, 532, 708, 1207, 1208, 1210, 1211 |
| \ExplSyntaxOn | 1, 2 |
| F | |
| \fontencoding | 1749 |
| \fontfamily | 269, 274, 275, 356, 363, 364, 380, 381, 393, 394, 412, 413, 427, 428, 440, 441, 456, 457, 471, 472, 485, 486, 1749 |
| \fontseries | 27, 1749 |
| \fontshape | 5, 27, 1748 |
| \fontsize | 364, 381, 394, 413, 428, 441, 457, 472, 486, 511, 513, 666, 669 |
| \FancyVerbSpace | 70 |
| \fi | 16, 23, 36, 281, 296, 305, 360, 371, 374, 377, 387, 390, 406, 409, 421, 424, 434, 437, 450, 453, 465, 468, 479, 482, 493, 496, 518, 528, 534, 557, 583, 584, 594, 595, 601, 605, 606, 616, 620, 624, 625, 632, 636, 647, 650, 656, 668, 682–684, 688–690, 735, 745, 796, 797, 800, 814, 821, 824, 842, 896–898, 903, 904, 913, 914, 940, 960, 984, 1058, 1069, 1092, 1124, 1128, 1166, 1178, 1214, 1222, 1230, 1238, 1240, 1241, 1280, 1284, 1289– 1291, 1419, 1426, 1481, 1751, 1752, 1781, 1803, 1840, 1841, 1857, 1863, 1873, 1877, 1888, 1895 |
| \fi: | 942, 946, 962, 968, 986, 995 |
| \font | 12, 15, 19, 364, 381, 394, 413, 428, 441, 457, 472, 486, 511, 513, 666, 669, 1148, 1181–1183, 1189, 1201, 1211, 1227 |
| \font@warning | 1860 |
| \fontdimen | 1157, 1171, 1175–1177, 1181–1183, 1188, 1189 |
| \fontfamily | 213, 259, 278 |
| \fontname | 667, 670 |
| \fontshape | 1742, 1753 |
| \fontspec | 24, 29, <u>211</u> |
| \fontspec_calc_scale:n | 1138, 1139, <u>1145</u> |
| \fontspec_check_lang:n | 948 |
| \fontspec_check_lang:nTF | 341, 447, 462, <u>948</u> , 1716, 1721 |
| \fontspec_check_ot_feat:n | 971 |
| \fontspec_check_ot_feat:nT | 799, 803, <u>970</u> |
| \fontspec_check_ot_feat:nTF | 403, 418, 851, <u>970</u> |
| \fontspec_check_script:n | 929 |
| \fontspec_check_script:nTF | 324, 431, <u>929</u> |
| \fontspec_complete_fontname:Nn | 1046, 1056, 1067, 1072, 1076, 1080, 1090, <u>1094</u> , 1133 |
| \fontspec_error:n | 54, 709, 1780 |
| \fontspec_error:nx | 55, 753, 1213, 1221, 1229, 1237 |
| \fontspec_fullname:n | 511, 513, 541, <u>570</u> , 666, 669, 699, 713 |
| \fontspec_get_features:n | 519, 696, 710, <u>747</u> |
| \fontspec_if_aat_feature:nn | 362 |
| \fontspec_if_aat_feature:nnTF | 1, <u>362</u> |
| \fontspec_if_current_language:n | 484 |
| \fontspec_if_current_language:nTF | 1, <u>484</u> |
| \fontspec_if_current_script:n | 470 |
| \fontspec_if_current_script:nTF | 1, <u>470</u> |
| \fontspec_if_detect_external:n | 560 |
| \fontspec_if_detect_external:nT | 505, 560 |
| \fontspec_if_feature:n | 392 |
| \fontspec_if_feature:nnn | 411 |
| \fontspec_if_feature:nnnTF | 1, <u>411</u> |
| \fontspec_if_feature:nTF | 1, <u>392</u> |
| \fontspec_if_fontspec_font: | 355 |
| \fontspec_if_fontspec_font:TF | 1, <u>355</u> |
| \fontspec_if_language:n | 439 |
| \fontspec_if_language:nn | 455 |
| \fontspec_if_language:nnTF | 1, <u>455</u> |
| \fontspec_if_language:nTF | 1, <u>439</u> |
| \fontspec_if_opentype: | 379 |
| \fontspec_if_opentype:TF | 1, <u>379</u> |
| \fontspec_if_script:n | 426 |
| \fontspec_if_script:nTF | 1, <u>426</u> |
| \fontspec_info:n | 59, 1152, 1886 |
| \fontspec_info:nx | 60, 672 |
| \fontspec_info:nxx | 61, 537 |
| \fontspec_init: | 501, <u>757</u> |
| \fontspec_iv_str_to_num:n | 416, 417, 461, <u>906</u> , 931, 950 |
| \fontspec_iv_str_to_num:w | 907, 909 |
| \fontspec_make_AAT_feature:nn | 820, 830 |

| | | |
|---|---|--|
| <code>\fontspec_make_AAT_feature_string:nn</code> 366, 793, 836, <u>882</u> | <code>\ifzf@configfile</code> 38, 1891 | |
| <code>\fontspec_make_feature:nnn</code> <u>816</u> , 863, 1418, 1425, 1475 | <code>\ifzf@math</code> 39, 1885 | |
| <code>\fontspec_make_ICU_feature:n</code> 823, 827, 845 | <code>\ifcase</code> 641 | |
| <code>\fontspec_namewrap:n</code> ... 571, 773, 1002 | <code>\ifcsname</code> 269, 294, 303, 356, 363, 380, 393, 412, 427, 440, 456, 471, 485, 521, 522, 536, 1749 | |
| <code>\fontspec_new_lang:nn</code> 335, 336, 338 | <code>\ifdefined</code> 1850 | |
| <code>\fontspec_new_script:nn</code> .. 318, 319, 321 | <code>\iffontchar</code> 19 | |
| <code>\fontspec_patch_fancyvrb:</code> 46, 68 | <code>\ifmmode</code> 36 | |
| <code>\fontspec_patch_listings:</code> 47, 73 | <code>\ifnum</code> ... 12, 645, 885, 934, 935, 954, 955, 977, 978, 1207, 1216, 1226, 1232 | |
| <code>\fontspec_patch_moreverb:</code> 45, 58 | <code>\ifodd</code> 888 | |
| <code>\fontspec_patch_verbatim:</code> 44, 49 | <code>\ifx</code> 367, 577, 588, 598, 608–610, 628, 629, 665, 671, 677, 680, 682, 688, 794, 837, 884, 894, 899, 913, 914, 1051, 1062, 1085, 1170, 1199, 1205, 1416, 1423, 1748, 1801, 1863, 1873, 1877 | |
| <code>\fontspec_select:nn</code> .. 212, 217, 222, 227, 233, 237, 241, 245, 256, 273, <u>499</u> | <code>\ifzf@atsui</code> 31, 365, 652, 792, 819 | |
| <code>\fontspec_set_bold:</code> 552, <u>575</u> | <code>\ifzf@firsttime</code> 26, 743, 812, 1124, 1128, 1164, 1287 | |
| <code>\fontspec_set_bold_italic:</code> .. 555, <u>603</u> | <code>\ifzf@graphite</code> 34 | |
| <code>\fontspec_set_bold_slanted:</code> .. 556, <u>627</u> | <code>\ifzf@icu</code> 32, 383, 396, 415, 430, 443, 459, 474, 488, 515, 654, 732, 798, 822, 1277, 1285, 1474 | |
| <code>\fontspec_set_font_dimen:NnN</code> 1148, 1149, 1155 | <code>\ifzf@math@euler</code> 35, 1805 | |
| <code>\fontspec_set_font_type:</code> . 382, 395, 414, 429, 442, 458, 473, 487, 512, <u>638</u> | <code>\ifzf@math@lucida</code> 36, 1810 | |
| <code>\fontspec_set_italic:</code> 553, <u>586</u> | <code>\ifzf@mm</code> 33, 1281, 1286 | |
| <code>\fontspec_set_slanted:</code> 554, <u>597</u> | <code>\ifzf@nobf</code> 27, 576, 605 | |
| <code>\fontspec_setkeys:xx</code> 46, 507, 509, 708, 751 | <code>\ifzf@noit</code> 28, 587, 606 | |
| <code>\fontspec_setkeys:xxx</code> 50, 516, 517 | <code>\ifzf@nosc</code> 29, 678 | |
| <code>\fontspec_trace:n</code> 62 | <code>\ifzf@package@euler@loaded</code> .. 37, 1777 | |
| <code>\fontspec_v_str_to_num:n</code> <u>906</u> , 974 | <code>\ifzf@tfm</code> 30 | |
| <code>\fontspec_warning:n</code> 56, 180, 280, 833, 848, 1256, 1260, 1288 | <code>\ignorespaces</code> 214, 282 | |
| <code>\fontspec_warning:nx</code> 57, 329, 346, 838, 855, 875, 1022, 1026, 1726 | <code>\InputIfFileExists</code> 1892 | |
| <code>\fontspec_warning:nxx</code> 58, 298, 307 | <code>\int_new:N</code> 44 | |
| <code>\frenchspacing</code> 13 | <code>\int_set:Nn</code> 397, 399, 445, 910 | |
| G | | |
| <code>\g@addto@macro</code> 207, 1200, 1209, 1219, 1227, 1235 | <code>\int_use:N</code> 544, 546 | |
| <code>\Gamma</code> 1821 | <code>\intexpr_compare:nTF</code> 1016 | |
| <code>\gdef</code> 33 | <code>\itdefault</code> 5, 590, 593, 612, 615, 619, 623, 682, 688, 722, 726, 1757, 1765, 1845, 1853, 1856 | |
| <code>\global</code> 62, 511, 513, 523, 526 | <code>\itshape</code> 8, 33, 34, <u>1755</u> | |
| <code>\grave</code> 1789 | K | |
| <code>\group_begin:</code> 1146 | <code>\key@ifundefined</code> . 297, 306, 870, 872, 874 | |
| <code>\group_end:</code> 1153 | <code>\keyval@alias@key</code> 314, <u>865</u> , 876, 878, 880, 1248 | |
| H | | |
| <code>\hat</code> 1795 | L | |
| <code>\hbox</code> 36 | <code>\l_fontspec_extension_tl</code> 571, 1007, 1012 | |
| <code>\hyphenchar</code> 12, 15, 1201, 1210, 1227 | <code>\l_fontspec_extensions_clist</code> . 352, 562 | |
| I | | |
| <code>\if@tempswa</code> 607, 942, 946, 962, 968, 986, 995 | | |

| | |
|--|----------------------------|
| <code>\l_fontspec_external_bool</code> | |
| 997, 1001, 1008 | |
| <code>\l_fontspec_lang_name_tl</code> | |
| 130, 517, 778, 1042 | |
| <code>\l_fontspec_lang_tl</code> .. 343, 402, 548, | |
| 734, 740, 779, 992, 1719, 1724, 1732 | |
| <code>\l_fontspec_mode_tl</code> 738, 781, 1027 | |
| <code>\l_fontspec_nfss_tl</code> .. 697, 704, 711, 720 | |
| <code>\l_fontspec_optical_size_tl</code> | |
| 573, 774, 1278, 1293 | |
| <code>\l_fontspec_pre_feat_tl</code> 541, 700, 714, <u>730</u> | |
| <code>\l_fontspec_rawfeatures_sclist</code> ... | |
| 541, 700, 714, 748, 755, 813 | |
| <code>\l_fontspec_renderer_tl</code> | |
| 572, 651, 653, 655, 775, 1019 | |
| <code>\l_fontspec_scale_tl</code> | |
| 173, 698, 712, 749, 1141–1143, 1151 | |
| <code>\l_fontspec_script_name_tl</code> | |
| 130, 140, 516, 776, 1037 | |
| <code>\l_fontspec_script_tl</code> 326, 401, | |
| 444, 460, 547, 733, 739, 777, 966, 992 | |
| <code>\l_fontspec_size_tl</code> . 706, 709, 712, 1131 | |
| <code>\l_fontspec_sizedfont_tl</code> 707, 713, 1133 | |
| <code>\l_fontspec_sizefeat_clist</code> | |
| 694, 705, 770, 1128 | |
| <code>\l_fontspec_strnum_int</code> | |
| 44, 327, 344, 416, 417, | |
| 461, 910, 935, 955, 979, 1717, 1722 | |
| <code>\l_tmpa_bool</code> | |
| 564, 566, 568 | |
| <code>\l_tmpa_num</code> | |
| 1013, 1016 | |
| <code>\l_tmpa_tl</code> | |
| 1013, 1019, 1027 | |
| <code>\Lambda</code> | |
| 1824 | |
| <code>\LaTeX</code> | |
| 21 | |
| <code>\latinencoding</code> | |
| 206, 209 | |
| <code>\leavevmode</code> | |
| 36 | |
| <code>\let</code> | |
| 8, 9, 33, 38, 70, | |
| 75, 234, 238, 242, 246, 267, 271, | |
| 284, 503, 504, 508, 510, 675, 748, | |
| 750, 789, 902, 1774, 1775, 1800, 1860 | |
| <code>\listing@line</code> | |
| 62 | |
| <code>\listingcont*(environment)</code> | |
| <u>58</u> | |
| <code>\loop</code> | |
| 934, 954, 977 | |
| <code>\lst@visiblespace</code> | |
| 75 | |
| <code>\LuaTeX</code> | |
| 21 | |
| <code>\luatex_if_engine:T</code> 17, 23, 198, 780, 1362 | |
| <code>\luatex_if_engine:TF</code> | |
| 8 | |
| <code>\luatexpostexhyphenchar</code> | |
| 785 | |
| <code>\luatexposthyphenchar</code> | |
| 783 | |
| <code>\luatexpreehyphenchar</code> | |
| 784 | |
| <code>\luatexprehyphenchar</code> 782, 1202, 1219, 1235 | |
| <code>\luatexRequireModule</code> | |
| 10 | |
| | M |
| <code>\mathalpha</code> | 1788–1797, 1811–1831 |
| <code>\mathbf</code> | 95, 1846, 1852 |
| <code>\mathbin</code> | 1832 |
| <code>\mathchardef</code> | 1799 |
| <code>\mathclose</code> | 1806, 1809, 1835, 1837 |
| <code>\mathdollar</code> | 1839 |
| <code>\mathit</code> | 95, 1756, 1845, 1853, 1856 |
| <code>\mathopen</code> | 1834, 1836 |
| <code>\mathord</code> | 1838, 1839 |
| <code>\mathpunct</code> | 1802, 1808 |
| <code>\mathrel</code> | 1807, 1833 |
| <code>\mathring</code> | 1797 |
| <code>\mathrm</code> | 1844, 1851, 1855 |
| <code>\mathsf</code> | 1847, 1858 |
| <code>\mathtt</code> | 1848, 1859 |
| <code>\mddefault</code> | 551, 590, 593, 600, |
| 1842–1845, 1847, 1848, 1851, 1853 | |
| <code>\msg_error:nn</code> | 12, 54 |
| <code>\msg_error:nnx</code> | 55 |
| <code>\msg_info:nn</code> | 59 |
| <code>\msg_info:nnx</code> | 60 |
| <code>\msg_info:nnxx</code> | 61 |
| <code>\msg_new:nnn</code> .. 3, 63, 75, 98, 103, 107, | |
| 111, 116, 121, 126, 132, 136, 142, | |
| 146, 150, 154, 159, 163, 167, 171, 175 | |
| <code>\msg_new:nnnn</code> | 68, 79, 87 |
| <code>\msg_redirect_module:nnn</code> | |
| 187, 188, 191, 192 | |
| <code>\msg_redirect_name:nnn</code> | 1257 |
| <code>\msg_trace:nn</code> | 62 |
| <code>\msg_warning:nn</code> | 56 |
| <code>\msg_warning:nnx</code> | 57 |
| <code>\msg_warning:nnxx</code> | 58 |
| <code>\multi@alias@key</code> | 312, <u>869</u> |
| | N |
| <code>\newAATfeature</code> | <u>293</u> |
| <code>\newcommand</code> | 662, 693, 742, |
| 771, 788, 811, 859, 862, 865, 869, 1745 | |
| <code>\newcount</code> | 40–43, 526 |
| <code>\newfontface</code> | <u>255</u> |
| <code>\newfontfamily</code> | <u>255</u> |
| <code>\newfontfeature</code> | <u>285</u> |
| <code>\newfontlanguage</code> | <u>333</u> , 1524–1714 |
| <code>\newfontscript</code> | <u>316</u> , 1483–1523 |
| <code>\newICUfeature</code> | <u>302</u> |
| <code>\newif</code> | 26–39 |
| <code>\newopentypefeature</code> | <u>302</u> |
| <code>\next</code> | 1800 |
| <code>\normalfont</code> | 219, 224, 229 |

| | |
|--|---|
| <code>\not@math@alphabet</code> | <code>\selectfont</code> 213, 259, 278, 1742, 1753 |
| 1741, 1756, 1760, 1764, 1768 | <code>\setboldmathrm</code> <u>232</u> |
| <code>\null</code> 36 | <code>\setkeys</code> 48, 52, 506, 860, 1003, |
| <code>\numexpr</code> 892 | 1009, 1036, 1041, 1258, 1415, 1422 |
| O | <code>\setlength</code> 1171, 1175–1177, 1188 |
| <code>\Omega</code> 1831 | <code>\setmainfont</code> 9, 27, 39, <u>216</u> , 231 |
| <code>\or</code> 643, 648 | <code>\SetMathAlphabet</code> 1844–1848, |
| P | 1851–1853, 1855, 1856, 1858, 1859 |
| <code>\PackageInfo</code> 755 | <code>\setmathrm</code> <u>232</u> |
| <code>\pagestyle</code> 16 | <code>\setmathsf</code> <u>232</u> |
| <code>\par</code> 24, 63 | <code>\setmathtt</code> <u>232</u> |
| <code>\Phi</code> 1829 | <code>\setmonofont</code> 11, 27, <u>216</u> |
| <code>\Pi</code> 1826 | <code>\setromanfont</code> <u>231</u> |
| <code>\pi</code> 38 | <code>\setsansfont</code> 10, 27, <u>216</u> |
| <code>\pm</code> 38 | <code>\SetSymbolFont</code> 1787, 1843, 1849 |
| <code>\prg_case_str:nnn</code> 1136 | <code>\sfdefault</code> 201, 223, 253 |
| <code>\prg_new_conditional:Nnn</code> | <code>\sffamily</code> 34 |
| ... 355, 362, 379, 392, 411, 426, | <code>\sidefault</code> 682, 688, |
| 439, 455, 470, 484, 560, 929, 948, 971 | 1739, 1742, 1757, 1761, 1765, 1769 |
| <code>\prg_return_false:</code> | <code>\Sigma</code> 1827 |
| 359, 368, 373, 376, 386, | <code>\sishape</code> <u>1739</u> |
| 389, 403, 405, 408, 418, 420, 423, | <code>\sldefault</code> 600, 631, 635, 725, 1761 |
| 431, 433, 436, 447, 449, 452, 462, | <code>\slshape</code> 1759, 1760 |
| 464, 467, 476, 478, 481, 490, 492, | <code>\space</code> 82, 173, 886, 889, 892 |
| 495, 568, 942, 946, 962, 968, 986, 995 | <code>\stepcounter</code> 286 |
| <code>\prg_return_true:</code> 357, 370, | <code>\string</code> 95, 100 |
| 384, 403, 418, 431, 447, 462, 476, | <code>\strip@pt</code> 1151 |
| 490, 568, 942, 946, 962, 968, 986, 995 | T |
| <code>\ProcessOptions</code> 195 | <code>\TeX</code> 21, 44 |
| <code>\providecommand</code> 1739 | <code>\textsf</code> 18, 20, 49 |
| <code>\Psi</code> 1830 | <code>\textsi</code> <u>1739</u> |
| Q | <code>\textvisiblespace</code> 20, 28 |
| <code>\q_nil</code> 907, 909 | <code>\the</code> 63, 287, 532, 1181–1183, 1189 |
| R | <code>\thelisting@line</code> 62 |
| <code>\relax</code> 36, 327, 344, | <code>\Theta</code> 1823 |
| 416, 417, 461, 892, 1201, 1202, | <code>\tilde</code> 1791 |
| 1219, 1227, 1235, 1717, 1722, | <code>\tl_clear:N</code> |
| 1733, 1741, 1760, 1764, 1768, 1799 | 704, 706, 749, 757–769, 774, 775, 1012 |
| <code>\repeat</code> 941, 961, 985 | <code>\tl_gset:cx</code> 538–540, 543, 545 |
| <code>\RequirePackage</code> 1, 2, 9, 196–198, 203 | <code>\tl_gset:Nx</code> 1151 |
| <code>\rmdefault</code> 200, 218, 252 | <code>\tl_gset_eq:cN</code> 547, 548 |
| <code>\rmfamily</code> 1147 | <code>\tl_if_empty:NF</code> 752 |
| S | <code>\tl_if_empty:NT</code> 651, 709, 807 |
| <code>\scdefault</code> 682, 688, 1757, 1761, 1765, 1769 | <code>\tl_if_empty:nTF</code> 831, 846 |
| <code>\sclist_put_right:Nn</code> <u>806</u> | <code>\tl_if_eq:nnTF</code> 498, 1255 |
| <code>\scshape</code> 33, <u>1755</u> | <code>\tl_if_eq:nvTF</code> 475, 489 |
| <code>\section</code> 18 | <code>\tl_if_eq:xxT</code> 722 |
| | <code>\tl_if_eq:xxTF</code> 5 |
| | <code>\tl_if_head_eq_charcode_p:nN</code> . 920, 921 |
| | <code>\tl_if_in:nnT</code> 565 |
| | <code>\tl_put_right:Nx</code> 711 |

| | | | |
|-------------------------------|--|---------------------------------|---|
| \tl_replace_all_in:Nnn | 1098 | \xetex_or luatex:nnn . | <u>20</u> , 199, 638, |
| \tl_replace_all_in:Nnx | 1096 | | 730, 929, 948, 970, 1155, 1200, 1249 |
| \tl_set:cn | 1030–1034 | \XeTeXcharglyph | 1207, 1226 |
| \tl_set:Nn | 199–202, | \XeTeXcountvariations | 645 |
| | 326, 343, 460, 653, 655, 730, 776– | \XeTeXfeaturename | 883 |
| | 779, 781, 1007, 1037, 1042, 1095, | \XeTeXfonttype | 641 |
| | 1131, 1278, 1293, 1719, 1724, 1732 | \XeTeXisexclusivefeature | 885 |
| \tl_set:Nv | 401, 402, 444, 1019, 1027 | \XeTeXOTcountfeatures | 973 |
| \tl_set:Nx | 352, 697, 1141, 1143 | \XeTeXOTcountlanguages | 951 |
| \tl_set_eq:NN | | \XeTeXOTcountscripts | 932 |
| | 204–206, 208, 209, 218, 223, 228, 707 | \XeTeXOTfeaturetag | 978 |
| \ttdefault | 202, 228, 254 | \XeTeXOTlanguage tag | 955 |
| \two@digits | 1425 | \XeTeXOTscripttag | 935 |
| \typeout | 1893, 1894 | \XeTeXselectorname | 886, 889, 892 |
| U | | \Xi | 1825 |
| \unless .. | 294, 303, 521, 536, 576, 587, | \XKV@rm .. | 509, 510, 710, 752, 753, 1416, 1423 |
| | 665, 678, 680, 794, 812, 884, 888, | \XKV@tfam .. | 113, 118, 123, 128, 1417, 1424 |
| | 894, 899, 1124, 1128, 1164, 1416, 1423 | \XKV@tkey | 113, 118, 123, 128 |
| \updefault | 551, | \xlx@defaultthyphenchar | 13, 17 |
| | 579, 582, 1769, 1842–1844, 1846– | \xxt@visiblespace | <u>18</u> , 33, 70, 75 |
| | 1849, 1851, 1852, 1855, 1858, 1859 | \xxt@visiblespace@fallback | 22, <u>25</u> |
| \upshape | 9, <u>1755</u> | \xxt@vprintspaces | <u>31</u> , 42, 52, 55, 65 |
| \Upsilon | 1828 | Z | |
| \use:c | 364, 398, 400, 446 | \z@ .. | 12, 933, 952, 975, 1208, 1218, 1226, 1234 |
| \use:n | 773 | \zap@space | 352, 531, 1091, 1125, 1129 |
| \use:x | 45, 48, 52, 257, 272, 718, 724 | \zf@e | 1165, 1168, 1204, 1243 |
| \use_i:nn | 16 | \zf@eii | 1165 |
| \use_ii:nn | 18 | \zf@eiii | 1165 |
| \use_none:n | 925 | \zf@adjust . | 720, 726, 750, 1179, 1180, |
| \usefont | 27 | | 1189, 1200, 1209, 1219, 1227, 1235 |
| \usepackage | 6, 7 | \zf@atsuifalse | 640 |
| \UTFencname | 204 | \zf@atsuitrue | 644 |
| V | | \zf@basefont . | 364, 381, 394, 413, 428, |
| \verb | 24, 27, 29, <u>35</u> , 39, 44–46 | | 441, 457, 472, 486, 511, 513, 641, |
| \verb* | <u>35</u> | | 645, 675, 883, 885, 886, 889, 892, |
| \verb@eol@error | 38 | | 932, 935, 951, 955, 973, 978, 1149, |
| verbatim* (environment) | <u>49</u> | | 1171, 1175–1177, 1188, 1207, 1226 |
| \verbatim@font | 39 | \zf@bf | 577, 581, 609, 618, 757, 1056 |
| \verbatim@line | 63 | \zf@bf@feat | 579, 582, 764, 1104 |
| \verbatim@processline | 61 | \zf@bfit | 608, 622, 759, 1072 |
| \verbatim@start | 52, 65 | \zf@bfit@feat .. | 612, 615, 619, 623, 766, 1112 |
| X | | \zf@bfsl | 628, 634, 761, 1080 |
| \xdef | 535, 744, 813 | \zf@bfsl@feat | 631, 635, 768, 1120 |
| \xetex_if_engine:F | 7 | \zf@check@one@char | 1204, 1243 |
| \xetex_if_engine:T ... | 1003, 1036, 1041 | \zf@DeclareFontShape . | 676, 681, 687, <u>693</u> |
| \xetex_if_engine:TF | 15, 20 | \zf@default@options | |
| \xetex_or luatex:nn .. | <u>15</u> , 790, 817, | | 165, 266, 267, 271, 507, 539 |
| | 1017, 1025, 1206, 1225, 1276, 1335 | \zf@define@feature@option | |
| | | | 300, 309, <u>859</u> , 1310– |
| | | | 1333, 1344–1352, 1354–1361, |

| | |
|--|---|
| 1363, 1364, 1367–1380, 1382–1392, 1394–1398, 1400, 1402–1408, 1410–1413, 1430–1440, 1442–1448, 1450–1457, 1459–1471 | \zf@mmfalse 640 |
| \zf@define@font@feature 295, 304, <u>859</u> , 1309, 1343, 1353, 1366, 1381, 1393, 1401, 1409, 1429, 1441, 1449, 1458, 1472 | \zf@mmtrue 646 |
| \zf@enc 199, 203–206, 208, 209, 549, 719, 725, 1842–1849, 1851–1853, 1855, 1856, 1858, 1859 | \zf@nobffalse 1055 |
| \zf@family 213, 218, 223, 228, 234, 238, 242, 246, 259, 278, 398, 400–402, 444, 446, 475, 489, 535, 536, 538– 540, 543, 545, 547–549, 719, 725, 726 | \zf@nobftrue 999, 1052 |
| \zf@family@long .. 503, 521, 530, 535, 744 | \zf@noitfalse 1066 |
| \zf@firsttimefalse 520 | \zf@noittrue 1000, 1063 |
| \zf@firsttimetrue 514 | \zf@noscfalse 1089 |
| \zf@font@feat 510, 519, 551, 579, 582, 590, 593, 600, 612, 615, 619, 623, 631, 635 | \zf@nosctrue 1086 |
| \zf@font@warning 1774, 1860 | \zf@package@euler@loadedfalse ... 1772 |
| \zf@fontname 77, 109, 114, 119, 124, 129, 134, 139, 173, 502–504, 508, 541, 550, 578, 589, 611, 674, 686, 699, 707, 1096 | \zf@package@euler@loadedtrue 1771 |
| \zf@graphitefalse 640 | \zf@rmboldmaths 238, 1850–1853 |
| \zf@icufalse 640, 772 | \zf@rmmaths 234, 252, 1842–1846, 1849, 1855, 1856 |
| \zf@icutrue 649, 660 | \zf@sc 677, 686, 762, 1090 |
| \zf@eit 588, 592, 610, 614, 758, 1067 | \zf@sc@feat 682, 688, 769, 1124 |
| \zf@eit@feat 590, 593, 765, 1108 | \zf@sfnaths 242, 253, 1847, 1858 |
| \zf@make@font@shapes 550, 578, 581, 589, 592, 599, 611, 614, 618, 622, 630, 634, <u>662</u> | \zf@sl 598, 599, 629, 630, 760, 1076 |
| \zf@make@smallcaps 679, <u>788</u> | \zf@sl@feat 600, 767, 1116 |
| \zf@math <u>1771</u> , 1887 | \zf@smallcaps 680, 681, 789, 795, 799, 803 |
| \zf@math@eulertrue 1778 | \zf@tfmfalse 640 |
| \zf@math@lucidatrue 1783–1785 | \zf@tfmtrue 642 |
| \zf@math@maybe <u>1861</u> | \zf@ttmaths 246, 254, 1848, 1859 |
| \zf@merge@shape <u>1745</u> , 1757, 1761, 1765, 1769 | \zf@up 504, 508, 511, 513, 1046 |
| | \zf@up@feat 551, 763, 1100 |
| | \zf@update@family 289, 325, 342, <u>742</u> , 840, 852, 1015, 1038, 1043, 1047, 1053, 1057, 1064, 1068, 1073, 1077, 1081, 1087, 1091, 1101, 1105, 1109, 1113, 1117, 1121, 1125, 1129, 1142, 1163, 1187, 1192, 1196, 1245, 1252, 1264, 1268, 1272, 1279, 1282, 1294, 1298, 1302, 1306, 1336, 1339, 1476, 1479, 1718, 1723, 1731, 1736 |
| | \zf@update@ff 290, <u>811</u> , 841, 853, 1193, 1246, 1253, 1265, 1269, 1273, 1283, 1299, 1303, 1307, 1337, 1340, 1477, 1480, 1737 |
| | \zf@wordspace@parse 1165, <u>1168</u> |