

The fontspec package

WILL ROBERTSON and KHALED HOSNY

2010/07/11 v2.0a

Contents

1 History	3	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	4	8.2 Scale	15
3.1 Maths fonts adjustments	4	8.3 Interword space	16
3.2 Configuration	4	8.4 Post-punctuation space	16
3.3 Warnings	5	8.5 The hyphenation character	17
I General font selection	5	8.6 Optical font sizes	17
4 Font selection	5	II OpenType	18
4.1 By font name	5	9 Introduction	18
4.2 By file name	6	10 Complete listing of OpenType font features	19
5 Default font families	7	10.1 Ligatures	19
6 New commands to select font families	7	10.2 Letters	19
6.1 More control over font shape selection	8	10.3 Numbers	20
6.2 Math(s) fonts	10	10.4 Contextuals	21
6.3 Miscellaneous font selecting details	10	10.5 Vertical Position	23
7 Selecting font features	11	10.6 Fractions	23
7.1 Default settings	11	10.7 StylisticSet	23
7.2 Changing the currently selected features	11	10.8 Alternates	24
7.3 Priority of feature selection	11	10.9 Style	25
7.4 Different features for different font shapes	12	10.10Diacritics	27
		10.11Kerning	27
		10.12Font transformations	27
		10.13Annotation	28
		10.14CJK shape	29
		10.15Character width	29
		10.16Vertical typesetting	31

10.17OpenType scripts and languages	31	17 Renaming existing features & options	42
III LuaTeX-only font features	31	18 Programming details	43
11 OpenType font feature files	33	VI The patching/improvement of L^AT_EX 2_ε and other packages	44
IV Fonts and features with X_YTeX	33	19 Inner emphasis	44
12 X _Y TeX-only font features	34	20 Unicode footnote symbols	44
12.1 Mapping	34	21 Verbatim	44
12.2 Letter spacing	34	22 Discretionary hyphenation: \-	45
12.3 Different font technologies: AAT and ICU	34	VII fontspec.sty	46
12.4 Optical font sizes	35	23 Implementation	46
13 Mac OS X's AAT fonts	35	23.1 Bits and pieces	46
13.1 Ligatures	36	23.2 Error/warning messages	47
13.2 Letters	36	23.3 Option processing	50
13.3 Numbers	36	23.4 Packages	51
13.4 Contextuals	36	23.5 Encodings	51
13.5 Vertical position	37	23.6 User commands	51
13.6 Fractions	37	23.7 Programmer's interface	56
13.7 Variants	38	23.8 Internal macros	60
13.8 Alternates	38	23.9 keyval definitions	75
13.9 Style	39	23.10Italic small caps	94
13.10CJK shape	39	23.11Selecting maths fonts	95
13.11Character width	39	23.12Finishing up	99
13.12Vertical typesetting	39	VIII fontspec.lua	100
13.13Diacritics	39	IX fontspec-patches.sty	103
13.14Annotation	40	23.13Unicode footnote symbols	103
14 AAT & Multiple Master font axes	40	23.14Emph	103
V Programming interface	40	23.15\-	103
15 Defining new features	41	23.16Verbatims	103
16 Going behind fontspec's back	41	X fontspec.cfg	106

1 History

This package began life as a \LaTeX interface to select system-installed Mac OS X fonts in Jonathan Kew's X_{\TeX} , the first widely-used Unicode extension to \TeX . Over time, X_{\TeX} was extended to support OpenType fonts and then was ported into a cross-platform program to run also on Windows and Linux.

More recently, $\text{Lua}\TeX$ 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 $\text{Con}\TeX\text{t Mk. IV}$ is a re-write of his powerful typesetting system, taking full advantage of $\text{Lua}\TeX$'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 \LaTeX when run with the $\text{Lua}\TeX$ 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_{\TeX} or $\text{Lua}\TeX$ to load OpenType fonts in a \LaTeX 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 \LaTeX , since \LaTeX '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_{\TeX} should have little or no difficulty switching over to $\text{Lua}\TeX$.

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

This manual for version 2 of `fontspec` is still in the process of being re-written. If you see any typeset examples that are broken, please let me know! I've managed to go over a lot of them but some have been brought over from the old version without being proofed.

I'd also like to reduce the number of non-free fonts used in these examples. If you know any freely available fonts that could be used as alternative to any of the fonts in this document, please suggest them to me. Finally, if any aspect of the documentation is unclear or you would like to suggest more examples that could be made, get in touch. (Contributions especially welcome.)

3 Package loading and options

For basic use, no package options are required:

```
\usepackage{fontspec}
```

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`, which is distributed with the luaotfload package. Note that if you do not execute this script beforehand, the first time you attempt to typeset the process will pause for (up to) several minutes. (But only the first time.) Please see the luaotfload 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 the `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 15 on page 41](#) and [Section 17](#)), 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.

¹An example is distributed with the package.

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.

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 Xe_{La}TeX-only general font features, see [Part IV on page 34](#)
- For Lua_{La}TeX-only general font features, see [Part III on page 33](#)
- For features for AAT fonts in Xe_{La}TeX, see [Part Section 13 on page 35](#)

4 Font selection

In both Lua_{La}TeX and Xe_{La}TeX, fonts can be selected either by 'font name' or by 'file name'.

4.1 By font name

Fonts known to Lua_{La}TeX or Xe_{La}TeX 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 \LaTeX 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 \XeTeX ; 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/ ,
```

Example 1: Loading the default, sans serif, and monospaced fonts.

	<code>\setmainfont{TeX Gyre Bonum}</code>
	<code>\setsansfont[Scale=MatchLowercase]{Latin Modern Sans}</code>
	<code>\setmonofont[Scale=MatchLowercase]{Inconsolata}</code>
Pack my box with five dozen liquor jugs.	<code>\rmfamily\pangram\par</code>
Pack my box with five dozen liquor jugs.	<code>\sffamily\pangram\par</code>
Pack my box with five dozen liquor jugs.	<code>\ttfamily\pangram</code>

```
UprightFont    = *-regular ,
BoldFont       = *-bold ,
... ]
{texgyrepagella}
```

Note that X_YTeX and LuaTeX 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`. See Example 1. 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.

Example 2: Defining new font families.	
This is a <i>note</i> .	<code>\newfontfamily\notefont{Kurier}</code> <code>\notefont This is a \emph{note}.</code>
Example 3: Defining a single font face.	
<i>where is all the vegemite</i>	<code>\newfontface\fancy</code> <code>[Contextuals={WordInitial,WordFinal}]</code> <code>{Hoefler Text Italic}</code> <code>\fancy where is all the vegemite</code> <code>% \emph, \textbf, etc., all don't work</code>

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, demonstrated in Example 2. This macro should be used to create commands that would be used in the same way as `\rmfamily`, for example. If you would like to create a command that only changes the font inside its argument (like `\mph`) define it using regular L^AT_EX commands:

```
\newcommand\textnote[1]{\{\notefont #1\}}
\textnote{This is a note.}
```

Note that the double braces are intentional; the inner pair are used to to delimit the scope of the font change.

`\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, shown in Example 3, which is repeated in Section 13.4 on page 36.

6.1 More control over font shape selection

```
BoldFont = <font name>
ItalicFont = <font name>
BoldItalicFont = <font name>
SlantedFont = <font name>
BoldSlantedFont = <font name>
SmallCapsFont = <font name>
```

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`

Example 4: Explicit selection of the bold font.

	<code>\fontspec[BoldFont={Helvetica Neue}]</code>
Helvetica Neue UltraLight	<code>{Helvetica Neue UltraLight}</code>
<i>Helvetica Neue UltraLight Italic</i>	<code>Helvetica Neue UltraLight \\\</code>
Helvetica Neue	<code>{\itshape Helvetica Neue UltraLight Italic} \\\</code>
<i>Helvetica Neue Italic</i>	<code>{\bfseries Helvetica Neue } \\\</code>
	<code>{\bfseries\itshape Helvetica Neue Italic} \\\</code>

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. See Example 4.

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. (This has been shown previously in this section.) For example, some space can be saved instead of writing ‘Baskerville SemiBold’:

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

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

```
\fontspec[UprightFont={* SemiBold},
      BoldFont={* Bold}]{Baskerville}
```

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 [Section 7.4](#) 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 L^AT_EX freezes the maths fonts after this stage of the processing. The `fontspec` package must also be loaded after any maths font packages (*e.g.*, `euler`) to be successful. (Actually, it is *only* `euler` that is the problem.²)

Note that you may find that loading some maths packages won't be as smooth as you expect since `fontspec` (and X_YL^AT_EX in general) breaks many of the assumptions of T_EX 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.

```
\setmathrm [⟨font features⟩] {⟨font name⟩}
\setmathsf [⟨font features⟩] {⟨font name⟩}
\setmathtt [⟨font features⟩] {⟨font name⟩}
\setboldmathrm [⟨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}
```

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:

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}}`

Example 5: A demonstration of the `\defaultfontfeatures` command.

	<code>\fontspec{TeX Gyre Adventor}</code>
	Some default text 0123456789 \
	<code>\defaultfontfeatures{</code>
	Numbers=OldStyle, Color=888888
	<code>}</code>
	<code>\fontspec{TeX Gyre Adventor}</code>
Some default text 0123456789	Now grey, with old-style figures:
Now grey, with old-style figures: 0123456789	0123456789

`\renewcommand\emshape{\bfseries}`
Nested emphasis is controlled by the `\emminershape` command. For example, for `\emph{\emph{...}}` to produce small caps:
`\renewcommand\emminershape{\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, shown in Example 5. New calls of `\defaultfontfeatures` overwrite previous ones.

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 Example 6.

`\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,

Example 6: A demonstration of the `\addfontfeatures` command.

<p>'In 1842, 999 people sailed 97 miles in 13 boats. In 1923, 111 people sailed 54 miles in 56 boats.'</p> <table border="1"> <thead> <tr> <th>Year</th> <th>People</th> <th>Miles</th> <th>Boats</th> </tr> </thead> <tbody> <tr> <td>1842</td> <td>999</td> <td>75</td> <td>13</td> </tr> <tr> <td>1923</td> <td>111</td> <td>54</td> <td>56</td> </tr> </tbody> </table>	Year	People	Miles	Boats	1842	999	75	13	1923	111	54	56	<pre> \fontspec[Numbers={Proportional,OldStyle}] {TeX Gyre Adventor} '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 @{}} Year & People & Miles & Boats \\ \hline 1842 & 999 & 75 & 13 \\ 1923 & 111 & 54 & 56 \end{tabular} </pre>
Year	People	Miles	Boats										
1842	999	75	13										
1923	111	54	56										

Example 7: Features for, say, just italics.

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

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. See Example 7.

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, as shown in Example 8.

Example 8: Multiple Master-like features in AAT fonts.

Skia	<code>\fontspec[BoldFont={Skia},</code>
Skia 'Bold'	<code>BoldFeatures={Weight=2}]{Skia}</code>
	<code>Skia \\\bfseries Skia 'Bold'</code>

Example 9: An example of setting the SmallCapsFeatures separately for each font shape.

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

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 somewhat ludicrous Example 9.

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

Example 10: An example of specifying different font features for different sizes of font with SizeFeatures.

	<code>\fontspec[SizeFeatures={</code>
	<code>{Size={-8}, Font=TeX Gyre Bonum Italic, Color=AA0000},</code>
<i>Small</i>	<code>{Size={8-14}, Color=00AA00},</code>
<i>Normal size</i>	<code>{Size={14-}, Color=0000AA}}]{TeX Gyre Chorus}</code>
<i>Large</i>	<code>{\scriptsize Small\par} Normal size\par {\Large Large\par}</code>

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

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

be used anyway. A demonstration to hopefully clarify these details is shown in Example 10. A less trivial example is shown in the context of optical font sizes in Section 8.6 on page 17.

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 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.) Transparency is supported by

Example 11: Selecting colour with transparency.



```
\fontsize{48}{48}
\fontspec{TeX Gyre Bonum Bold}
{\addfontfeature{Color=FF000099}W}\kern-1ex
{\addfontfeature{Color=0000FF99}S}\kern-0.8ex
{\addfontfeature{Color=DDBB2299}P}\kern-0.8ex
{\addfontfeature{Color=00BB3399}R}
```

X_YLaTeX with the xdv2pdf driver and with Lua_YTeX; X_YLaTeX with the xdvipdfmx driver is not supported.

If you load the xcolor package, you may use any named color instead of writing the colours in hexadecimal.

```
\usepackage{xcolor}
...
\fontspec[Color=red]{Verdana} ...
\definecolor{Foo}{rgb}{0.3,0.4,0.5}
\fontspec[Color=Foo]{Verdana} ...
```

The color package is *not* supported; use xcolor instead.

You may specify the transparency with a named font using the Opacity feature:

```
\fontspec[Color=red,Opacity=0.7]{Verdana} ...
```

The Opacity feature may only be used in conjunction with the Color feature; it will be silently ignored if it appears by itself.

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 7](#). 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; these features are shown in [Example 12](#).

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.

Example 12: Automatically calculated scale values.

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

Example 13: Scaling the default interword space. An exaggerated value has been chosen to emphasise the effects here.

	<code>\fontspec{TeX Gyre Termes}</code>
	Some text for our example to take up some space, and to demonstrate the default interword space.
Some text for our example to take up some space, and to demonstrate the default interword space.	<code>\bigskip</code>
	<code>\addfontfeature{ WordSpace = 0.3 }</code>
Sometextforourexampletotakeupsomespace,andtodemonstrate the default interword space.	Some text for our example to take up some space, and to demonstrate the default interword space.

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, 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 default value, or a triplet of comma-separated values to scale the nominal value, the stretch, and the shrink of the interword space by, respectively. (`WordSpace={x}` is the same as `WordSpace={x,x,x}`.)

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; this is demonstrated in Example 14. Note that `PunctuationSpace=0` is *not* equivalent to `\frenchspacing`, although the difference will only be apparent when a line of text is under-full.

Example 14: Scaling the default post-punctuation space.	
	<code>\nonfrenchspacing</code>
	<code>\fontspec{TeX Gyre Schola}</code>
	<code>Letters, Words. Sentences. \par</code>
Letters, Words. Sentences.	<code>\fontspec[PunctuationSpace=2]{TeX Gyre Schola}</code>
Letters, Words. Sentences.	<code>Letters, Words. Sentences. \par</code>
Letters, Words. Sentences.	<code>\fontspec[PunctuationSpace=0]{TeX Gyre Schola}</code>
	<code>Letters, Words. Sentences.</code>

Example 15: Explicitly choosing the hyphenation character.	
	<code>\def\text{\fbox{\parbox{1.55cm}{%</code>
	<code>EXAMPLE HYPHENATION%</code>
	<code>}}\quad\quad\null\par\bigskip}</code>
<div style="border: 1px solid black; padding: 2px; display: inline-block;">EXAMPLE HYPHENATION</div>	<code>\fontspec{Linux Libertine}</code>
	<code>\addfontfeature{HyphenChar=None}</code>
	<code>\text</code>
<div style="border: 1px solid black; padding: 2px; display: inline-block;">EXAMPLE HYPHEN+ ATION</div>	<code>\addfontfeature{HyphenChar={+}}</code>
	<code>\text</code>

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.

This package redefines L^AT_EX'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_YL^AT_EX *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, as shown in Example 16.

Automatic OpenType optical scaling is in Example 17, in which we've scaled down some large text in order to be able to compare the difference for equiva-

Example 16: Optical size substitution is suppressed when set to zero.		
	<code>\fontspec[OpticalSize=0]{Warnock Pro Caption}</code>	
	Warnock Pro optical sizes	<code>\</code>
	<code>\fontspec[OpticalSize=0]{Warnock Pro}</code>	
	Warnock Pro optical sizes	<code>\</code>
Warnock Pro optical sizes	<code>\fontspec[OpticalSize=0]{Warnock Pro Subhead}</code>	
Warnock Pro optical sizes	Warnock Pro optical sizes	<code>\</code>
Warnock Pro optical sizes	<code>\fontspec[OpticalSize=0]{Warnock Pro Display}</code>	
Warnock Pro optical sizes	Warnock Pro optical sizes	

Example 17: A demonstration of automatic optical size selection.		
	<code>\fontspec{Warnock Pro}</code>	
	Automatic optical size	<code>\</code>
Automatic optical size	<code>\scalebox{0.4}{\Huge</code>	
Automatic optical size	Automatic optical size}	

lent font sizes (this gives the same output as we saw in the previous example for Warnock Pro Display).

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}
```

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

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

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.

Example 18: An example of the Ligatures feature.

strict → strict	<pre> \def\test#1#2{% #2 \$\to\$ {\addfontfeature{#1} #2}\} \fontspec{Linux Libertine} \test{Ligatures=Historical}{strict} \test{Ligatures=Rare}{wurtzite} \test{Ligatures=NoCommon}{firefly} </pre>
wurtzite → wurtzite	
firefly → firefly	

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 5](#) 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](#). A demonstration with the Linux Libertine fonts³ is shown in [Example 18](#).

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.

³<http://www.linuxlibertine.org/>

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

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

Example 19: Small caps from lowercase or uppercase letters.

	<code>\fontspec[Letters=SmallCaps]{TeX Gyre Adventor}</code>
THIS SENTENCE NO VERB	THIS SENTENCE no verb
	<code>\fontspec[Letters=UppercaseSmallCaps]{TeX Gyre Adventor}</code>
THIS SENTENCE NO verb	THIS SENTENCE no verb

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’). This difference is shown in Example 19. ‘Unicase’ is a weird hybrid of upper and lower case letters.

Note that The `Uppercase` option will (probably) not actually map letters to uppercase.⁴ It is designed select various uppercase forms for glyphs such as accents and dashes, such as shown in Example 20.

The Kerning feature also contains an `Uppercase` option, which adds a small amount of spacing in between letters (see [Section 10.11 on page 27](#)).

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 11](#).

⁴If you want automatic uppercase letters, look to L^AT_EX’s `\MakeUppercase` command.

Example 20: An example of the `Uppercase` option of the `Letters` feature.

	<code>\fontspec{Linux Libertine}</code>
UPPER-CASE example	UPPER-CASE example
	<code>\addfontfeature{Letters=Uppercase}</code>
UPPER-CASE example	UPPER-CASE example

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

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

Example 21: The effect of the SlashedZero option.

	<code>\fontspec[Numbers=Lining]{TeX Gyre Bonum}</code>
	0123456789
0123456789 0123456789	<code>\fontspec[Numbers=SlashedZero]{TeX Gyre Bonum}</code>
	0123456789

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’, shown in Example 21.

The Arabic option maps numerals to their Arabic or Persian equivalents based on the current Language setting (see [Section 10.17 on page 31](#)), shown in Example 22 using the Zar fonts⁵. This option is based on a LuaTeX feature of the luatload package, not an OpenType feature. (Thus, this feature is unavailable in XeTeX.)

10.4 Contextuals

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

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.

Example 22: An example of number remapping to Arabic or Persian. (LuaTeX only.)

• ١٢٣٤٥٦٧٨٩	<code>\fontspec[Script=Arabic,Numbers=Arabic]{XB Zar}</code>
	<code>{\addfontfeature{Language=Arabic}</code>
	<code>0123456789} \\\</code>
• ۱۲۳۴۵۶۷۸۹	<code>{\addfontfeature{Language=Parsi}</code>
	<code>0123456789}</code>

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

Feature	Option	Tag
Contextuals =	Swash	csw
	Alternate	calt
	WordInitial	init
	WordFinal	fina
	LineFinal	falt
	Inner	medi

Example 23: An example of the Swashes option of the Contextuals feature.

<i>Without Contextual Swashes</i>	<code>\fontspec{Warnock Pro} \itshape</code>	
<i>With Contextual Swashes; cf. W C S</i>	<code>\fontspec[Contextuals=Swash]{Warnock Pro}</code>	<code>\\</code>

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

Feature	Option	Tag
VerticalPosition =	Superior	sup
	Inferior	sub
	Numerator	numr
	Denominator	dn
	ScientificInferior	sinf
	Ordinal	ordn

Example 24: The VerticalPosition feature.

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

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

Feature	Option	Tag
Fractions	= On	frac
	Alternate	afrc

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 punctuation, 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. These are all shown in Example 24

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

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 automatically.

10.6 Fractions

For OpenType fonts use a regular text slash to create fractions, but the Fraction feature must be explicitly activated. Some (Asian fonts predominantly) also provide for the Alternate feature. These are both shown in Example 25.

10.7 StylisticSet

This feature selects a ‘Stylistic Set’ variation, which usually corresponds to an alternate glyph style for a range of characters (usually an alphabet or subset thereof). This feature is specified numerically. These correspond to OpenType features ss01, ss02, etc.

⁵http://wiki.irmug.org/index.php/X_Series_2

Example 25: The Fractions feature.

$\frac{1}{2}$	$\frac{1}{4}$	$\frac{5}{6}$	13579/24680	<code>\fontspec{Hiragino Maru Gothic Pro W4}</code>
$\frac{1}{2}$	$\frac{1}{4}$	$\frac{5}{6}$	13579/24680	<code>1/2 \quad 1/4 \quad 5/6 \quad 13579/24680 \\\</code>
$\frac{1}{2}$	$\frac{1}{4}$	$\frac{5}{6}$	13579/24680	<code>\addfontfeature{Fractions=On}</code>
$\frac{1}{2}$	$\frac{1}{4}$	$\frac{5}{6}$	13579/24680	<code>1/2 \quad 1/4 \quad 5/6 \quad 13579/24680 \\\</code>
$\frac{1}{2}$	$\frac{1}{4}$	$\frac{5}{6}$	13579/24680	<code>\addfontfeature{Fractions=Alternate}</code>
$\frac{1}{2}$	$\frac{1}{4}$	$\frac{5}{6}$	13579/24680	<code>1/2 \quad 1/4 \quad 5/6 \quad 13579/24680 \\\</code>

Example 26: Insular (ancient) letterforms for the Junicond font, accessed with the StylisticSet feature.

Insular forms.	<code>\fontspec{Junicond}</code>
Injular popmj.	<code>Insular forms. \\\</code>
	<code>\addfontfeature{StylisticSet=2}</code>
	<code>Insular forms. \\\</code>

Two demonstrations from the Junicond font⁶ are shown in Example 26 and Example 27; thanks to Adam Buchbinder for the suggestion.

(This is a synonym of the Variant feature for AAT fonts.) See [Section 15 on page 41](#) for a way to assign names to stylistic sets, which should be done on a per-font basis.

10.8 Alternates

The Alternate feature (for the raw OpenType feature salt) is used to access alternate font glyphs when variations exist in the font, such as in Example 28. It uses a numerical selection, starting from zero, that will be different for each font. Note that the Style=Alternate option is equivalent to Alternate=0 to access the default case.

See [Section 15 on page 41](#) for a way to assign names to alternates, which must be done on a per-font basis.

Example 27: Enlarged minuscules (capital letters remain unchanged) for the Junicond font, accessed with the StylisticSet feature.

ENLARGED Minuscules.	<code>\fontspec{Junicond}</code>
ENLARGED Minuscules.	<code>ENLARGED Minuscules. \\\</code>
	<code>\addfontfeature{StylisticSet=6}</code>
	<code>ENLARGED Minuscules. \\\</code>

Example 28: The Alternate feature.	
A & h	<code>\fontspec{Linux Libertine}</code>
A & h	<code>\textsc{a} \& h \\\</code>
A & h	<code>\addfontfeature{Alternate=0}</code>
	<code>\textsc{a} \& h</code>

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

Feature Option	Tag
Style = Alternate	<code>salt</code>
Italic	<code>ital</code>
Ruby	<code>ruby</code>
Swash	<code>swsh</code>
Historic	<code>hist</code>
TitlingCaps	<code>titl</code>
HorizontalKana	<code>hkna</code>
VerticalKana	<code>vkna</code>

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.

Example 29 and Example 30 both contain glyph substitutions with similar characteristics. Note the occasional inconsistency with which font features are labelled; a long-tailed ‘Q’ could turn up anywhere!

In other features, larger breadths of changes can be seen, covering the style of an entire alphabet. See Example 31 and Example 32; in the latter, the `Italic` option affects the Latin text and the `Ruby` option the Japanese.

Note the difference here between the default and the horizontal style kana in Example 33.

⁶<http://junicode.sf.net>

Example 29: Example of the Alternate option of the Style feature.	
K Q R k v w y	<code>\fontspec{Warnock Pro}</code>
K Q R k v w y	<code>K Q R k v w y \\\</code>
K Q R k v w y	<code>\addfontfeature{Style=Alternate}</code>
	<code>K Q R k v w y</code>

Example 30: Example of the Historic option of the Style feature.

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

Example 31: Example of the TitlingCaps option of the Style feature.

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

Example 32: Example of the Italic and Ruby options of the Style feature.

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

Example 33: Example of the HorizontalKana and VerticalKana options of the Style feature.

ようこそ ワカヨタレソ	<code>\fontspec{Hiragino Mincho Pro}</code>
ようこそ ワカヨタレソ	<code>\kana</code>
ようこそ ワカヨタレソ	<code>{\addfontfeature{Style=HorizontalKana}</code>
	<code>\kana }</code>
ようこそ ワカヨタレソ	<code>{\addfontfeature{Style=VerticalKana}</code>
	<code>\kana }</code>

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

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

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

* This feature is activated by default.

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 10.2 on page 19](#), the Uppercase option will add a small amount of tracking between uppercase letters, seen in [Example 34](#).

10.12 Font transformations

In rare situations users may want to mechanically distort the shapes of the glyphs in the current font such as shown in [Example 35](#). Please don’t overuse these features; they are *not* a good alternative to having the real shapes.

Example 34: Adding extra kerning for uppercase letters.

UPPER-CASE EXAMPLE	<code>\fontspec{Warnock Pro}</code>
UPPER-CASE EXAMPLE	<code>UPPER-CASE EXAMPLE \</code>
UPPER-CASE EXAMPLE	<code>\addfontfeature{Kerning=Uppercase}</code>
UPPER-CASE EXAMPLE	<code>UPPER-CASE EXAMPLE</code>

Example 35: Artificial font transformations.		
		<code>\fontspec{Charis SIL} \emph{ABCxyz} \quad</code> <code>\fontspec[FakeSlant=0.2]{Charis SIL} ABCxyz</code>
		<code>\fontspec{Charis SIL} ABCxyz \quad</code> <code>\fontspec[FakeStretch=1.2]{Charis SIL} ABCxyz</code>
<i>ABCxyz</i>	<i>ABCxyz</i>	
ABCxyz	ABCxyz	<code>\fontspec{Charis SIL} \textbf{ABCxyz} \quad</code> <code>\fontspec[FakeBold=1.5]{Charis SIL} ABCxyz</code>
ABCxyz	ABCxyz	

Example 36: Annotation forms for OpenType fonts.		
1 2 3 4 5 6 7 8 9		
(1) (2) (3) (4) (5) (6) (7) (8) (9)		
① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨		
⓪ ① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨		
① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨		
① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨		
① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨		
① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨		
1. 2. 3. 4. 5. 6. 7. 8. 9.		
	<code>\fontspec{Hiragino Maru Gothic Pro}</code> <code>1 2 3 4 5 6 7 8 9</code> <code>\def\x#1{\{\addfontfeature{Annotation=#1}</code> <code>1 2 3 4 5 6 7 8 9 }\}}</code> <code>\x0\x1\x2\x3\x4\x5\x6\x7\x8\x9</code>	

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

If you want the bold shape to be faked automatically, or the italic shape to be slanted automatically, use the `AutoFakeBold` and `AutoFakeSlant` features. For example, the following two invocations are equivalent:

```
\fontspec[AutoFakeBold=1.5]{Charis SIL}
\fontspec[BoldFeatures={FakeBold=1.5}]{Charis SIL}
```

If both of the `AutoFake...` features are used, then the bold italic font will also be faked.

Currently, `FakeStretch` doesn't work in LuaTeX and will be ignored silently.

10.13 Annotation

Some fonts are equipped with an extensive range of numbers and numerals in different forms. These are accessed with the `Annotation` feature (OpenType feature `nalt`), selected numerically as shown in Example 36.

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

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

Example 37: Different standards for CJK ideograph presentation.

啞嚙軀 妍并訝	<code>\fontspec{Hiragino Mincho Pro}</code>
	<code>{\addfontfeature{CJKShape=Traditional}}</code>
	<code>\text }</code> <code>\\</code>
啞嚙軀 妍并訝	<code>{\addfontfeature{CJKShape=NLC}}</code>
	<code>\text }</code> <code>\\</code>
啞嚙軀 妍并訝	<code>{\addfontfeature{CJKShape=Expert}}</code>
	<code>\text }</code>

10.14 CJK shape

There have been many standards for how CJK ideographic glyphs are ‘supposed’ to look. Some fonts will contain many alternate glyphs available in order to be able to display these glyphs correctly in whichever form is appropriate. Both AAT and OpenType fonts support the following CJKShape options: Traditional, Simplified, JIS1978, JIS1983, JIS1990, and Expert. OpenType also supports the NLC option.

10.15 Character width

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

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.

The same situation occurs with numbers, which are provided in increasingly illegible compressed forms seen in `exrefcharwd`.

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

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

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

Example 38: Proportional or fixed width forms.

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

Example 39: Numbers can be compressed significantly.

	<pre> \fontspec[Renderer=AAT]{Hiragino Mincho Pro} {\addfontfeature{CharacterWidth=Full} ---12321---}\\ {\addfontfeature{CharacterWidth=Half} ---1234554321---}\\ {\addfontfeature{CharacterWidth=Third} ---123456787654321---}\\ {\addfontfeature{CharacterWidth=Quarter} ---12345678900987654321---} </pre>
— 1 2 3 2 1 —	
-1234554321-	
-123456787654321-	
-12345678900987654321-	

10.16 Vertical typesetting

TODO!

10.17 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 and Table 14 on the next page, 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 12.3 on page 34.

10.17.1 Defining new scripts and languages

`\newfontscript` Further scripts and languages may be added with the `\newfontscript` and `\newfontlanguage`
`\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 15 on page 41) is the error-checking that is performed when the script or language is requested.

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

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 14: Defined Languages for OpenType fonts. Aliased names are shown in adjacent positions marked with red pilcrows (⌵).

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

Figure 1: Addition of ligatures to a font via the FeatureFile font feature.

```
lookup fligatures {
  lookupflag 0;
  sub \f \i by \fi;
  sub \f \l by \fl;
} fligatures;

feature liga {
  script DFLT;
  language dflt ;
  lookup fligatures;

  script latn;
  language dflt ;
  lookup fligatures;
} liga;
```

Part III

LuaT_EX-only font features

11 OpenType font feature files

An OpenType font feature file is a plain text file describing OpenType layout feature of a font in a human-readable format. The syntax of OpenType feature files is defined by Adobe⁷.

Feature files can be used to add or customise OpenType feature of a font on the fly without editing the font file itself.

Feature files are loaded by passing its name or path to `FeatureFile`, then OpenType features defined in the file can be applied as usual.

For example, to add OpenType mapping of f-ligatures in Times New Roman font, one can define a `times-nr.fea` file with content shown in Figure 1. Then the font with the new ligatures can be loaded with:

```
\setmainfont[FeatureFile=times-nr.fea]{Times New Roman}
```

⁷http://www.adobe.com/devnet/opentype/afdko/topic_feature_file_syntax.html

Example 40: X_YTeX’s Mapping feature.

“¡A small amount of—text!”

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

Example 41: The LetterSpace feature.

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

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

Part IV

Fonts and features with X_YTeX

12 X_YTeX-only font features

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

12.1 Mapping

Mapping enables a X_YTeX text-mapping scheme, shown in Example 40.

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_YTeX documents.

12.2 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, shown in Example 41.

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.

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 10.2 on page 19).

12.3 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 10.17 on page 31](#) 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.*

12.4 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 14 on page 40](#) 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                \\\
```

13 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_YL^AT_EX (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_YL^AT_EX 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 14](#).

Example 42: Contextual glyph for the beginnings and ends of words.

	<code>\newfontface\fancy</code>
	<code>[Contextuals={WordInitial,WordFinal}]</code>
	<code>{Hoefler Text Italic}</code>
<i>~where is all the ~vegemite</i>	<code>\fancy where is all the vegemite</code>

Example 43: A contextual feature for the ‘long s’ can be convenient as the character does not need to be marked up explicitly.

	<code>\fontspec[Contextuals=Inner]{Hoefler Text}</code>
‘Inner’ fwashes can <i>fometimes</i>	<code>‘Inner’ swashes can \emph{sometimes} \</code>
contain the archaic long s.	<code>contain the archaic long~s.</code>

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

13.2 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.

13.3 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 11](#).

13.4 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 (Example 42), LineInitial, LineFinal, and Inner (Example 43, also called ‘non-final’ sometimes). As non-exclusive selectors, like the ligatures, you can turn them off by prefixing their name with No.

Example 44: Vertical position for AAT fonts.	
	<code>\fontspec{Skia}</code>
	Normal
	<code>\fontspec[VerticalPosition=Superior]{Skia}</code>
	Superior
	<code>\fontspec[VerticalPosition=Inferior]{Skia}</code>
	Inferior
	<code>\fontspec[VerticalPosition=Ordinal]{Skia}</code>
Normal superior inferior 1 st 2 nd 3 rd 4 th 0 th 8abcde	1st 2nd 3rd 4th 0th 8abcde

Example 45: Fractions in AAT fonts.	
	<code>\fontspec[Fractions=On]{Skia}</code>
	12 \quad 56 \quad \% fraction slash
	1/2 \quad 5/6 \quad \% regular slash
$\frac{1}{2}$ 5/6 1/2 5/6 13579/24680 13579/24680	<code>\fontspec[Fractions=Diagonal]{Skia}</code> 1357924680 \quad \% fraction slash \quad 13579/24680 \quad \% regular slash

13.5 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. These are shown in Example 44.

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

13.6 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. See Example 45.

Using the `Diagonal` option (AAT only), the font will attempt to create the fraction from superscript and subscript characters.

Some (Asian fonts predominantly) also provide for the `Alternate` feature shown in Example 46.

Example 46: Alternate design of pre-composed fractions.

$\frac{1}{2}$	$\frac{1}{4}$	$\frac{5}{6}$	13579/24680	<code>\fontspec{Hiragino Maru Gothic Pro}</code>
$\frac{1}{2}$	$\frac{1}{4}$	$\frac{5}{6}$	13579/24680	<code>1/2 \quad 1/4 \quad 5/6 \quad 13579/24680 \quad \backslash</code>
				<code>\addfontfeature{Fractions=Alternate}</code>
				<code>1/2 \quad 1/4 \quad 5/6 \quad 13579/24680</code>

Example 47: Nine variants of Zapfino.



```

\newcounter{var}\newcounter{trans}
\whiledo{\value{var}9}{%
  \stepcounter{trans}%
  \edef\1{%
    \noexpand\fontspec[Variant=\thevar,
      Color=005599\thetrans\thetrans]{Zapfino}}\1%
  \makebox[0.75\width]{d}%
  \stepcounter{var}}

```

13.7 Variants

The Variant feature takes a single numerical input for choosing different alphabetic shapes. Don't mind my fancy Example 47 :) I'm just looping through the nine (!) variants of Zapfino.

See [Section 15 on page 41](#) for a way to assign names to variants, which should be done on a per-font basis.

13.8 Alternates

Selection of Alternates *again* must be done numerically; see Example 48. See [Section 15 on page 41](#) for a way to assign names to alternates, which should be done on a per-font basis.

Example 48: Alternate shape selection must be numerical.

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

共産主義者は

共
産
主
義
者
は

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

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

13.9 Style

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

Typical examples for these features are shown in [Section 10.9](#).

13.10 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.

13.11 Character width

See [Section 10.15 on page 29](#) for relevant examples; the features are the same between OpenType and AAT fonts. AAT also allows CharacterWidth=Default to return to the original font settings.

13.12 Vertical typesetting

TODO: improve!

X_YTeX provides for vertical typesetting simply with the ability to rotate the individual glyphs as a font is used for typesetting, as shown in [Example 49](#).

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 X_YTeX documentation.

13.13 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.

⁸‘Ruby’ refers to a small optical size, used in Japanese typography for annotations.

Example 50: Various annotation forms.		
	<code>\fontspec{Hei Regular}</code>	
	1 2 3 4 5 6 7 8 9	<code>\</code>
	<code>\fontspec[Annotation=Circle]{Hei Regular}</code>	<code>\</code>
	1 2 3 4 5 6 7 8 9	<code>\</code>
1 2 3 4 5 6 7 8 9	<code>\fontspec[Annotation=Parenthesis]{Hei Regular}</code>	
① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨	1 2 3 4 5 6 7 8 9	<code>\</code>
(1) (2) (3) (4) (5) (6) (7) (8) (9)	<code>\fontspec[Annotation=Period]{Hei Regular}</code>	
1. 2. 3. 4. 5. 6. 7. 8. 9.	1 2 3 4 5 6 7 8 9	

Example 51: Continuously variable font parameters. These fonts are unfortunately quite rare.

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

Some fonts include *0/ etc.* as diacritics for writing Ø. You’ll want to turn this feature off (imagine typing *hello/goodbye* and getting ‘*helløgoodbye*’ instead!) by decomposing the two characters in the diacritic into the ones you actually want. I would recommend using the proper \TeX input conventions for obtaining such characters instead.

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

13.14 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 (see Example 50) with the following options: `Off`, `Box`, `RoundedBox`, `Circle`, `BlackCircle`, `Parenthesis`, `Period`, `RomanNumerals`, `Diamond`, `BlackSquare`, `BlackRoundSquare`, and `DoubleCircle`.

14 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 the demonstration in Example 51. Variations along a multiple master font’s optical size axis has been shown previously in Section 8.6 on page 17.


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

Part V

Programming interface

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

15 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. See Example 52.

`\newICUfeature` New OpenType features may be created with this command:

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

The synonym `\newopentypefeature` is provided for LuaLaTeX users.

Here's what it would look like in practise:

```
\newopentypefeature{Style}{NoLocalForms}{-loc1}
```

`\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 some like that shown in Example 53

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

16 Going behind fontspec's back

Expert users may wish not to use fontspec's feature handling at all, while still taking advantage of its LaTeX font selection conveniences. The `RawFeature` font feature

Example 53: Assigning new arbitrary features.

<i>sockdolager rubdown</i>	<code>\newfontfeature{AvoidD}{Special=Avoid d-collisions}</code>
<i>sockdolager rubdown</i>	<code>\newfontfeature{NoAvoidD}{Special=!Avoid d-collisions}</code>
	<code>\fontspec[AvoidD,Variant=1]{Zapfino}</code>
	<code>sockdolager rubdown \</code>
	<code>\fontspec[NoAvoidD,Variant=1]{Zapfino}</code>
	<code>sockdolager rubdown</code>

Example 54: Using raw font features directly.

PAGELLA SMALL CAPS	<code>\fontspec[RawFeature=+smcp]{TeX Gyre Pagella}</code>
	Pagella small caps

allows literal X_YTeX font feature selection when you happen to have the OpenType feature tag memorised.

Multiple features can either be included in a single declaration:

`[RawFeature=+smcp;+onum]`

or with multiple declarations:

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

17 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, such as shown in Example 55.

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

`\aliasfontfeatureoption` If you wish to change the name of a font feature option, it can be aliased to another with the command `\aliasfontfeatureoption{⟨font feature⟩}{⟨existing name⟩}{⟨new name⟩}`, such as shown in Example 56.

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,

Example 55: Renaming font features.

	<code>\aliasfontfeature{ItalicFeatures}{IF}</code>
	<code>\fontspec[IF = {Alternate=1}]{Hoefler Text}</code>
Roman Letters <i>And Swash</i>	Roman Letters \itshape And Swash

```

\aliasfontfeature{VerticalPosition}{Vert Pos}
\aliasfontfeatureoption{VerticalPosition}{ScientificInferior}{Sci Inf}
\fontspec[Vert Pos=Sci Inf]{Linux Libertine}
Scientific Inferior: 12345    Scientific Inferior: 12345

```

the `\newfontfeature` command should be used to declare a new, *e.g.*, `PurpleColor` feature:

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

Except that this example was written before support for named colours was implemented. But you get the idea.

18 Programming details

In some cases, it is useful to know what the \LaTeX font family of a specific `fontspec` font is. After a `\fontspec`-like command, this is stored inside the `\zf@family` macro. Otherwise, \LaTeX 's own `\f@family` macro can be useful here, too. The raw \TeX 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 <code>AAT</code> feature (#1,#2).
<code>\fontspec_if_opentype:TF</code>	Test whether the currently selected font is an OpenType font. Always true for <code>Lu</code> - <code>aTeX</code> 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>	Test 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 language tag #1. E.g.: <code>\fontspec_if_language:nTF {ROM} {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 language tag #2 in script #1. E.g.: <code>\fontspec_if_language:nnTF {cyr1} {SRB} {True} {False}</code> . Returns false if the font is not loaded by fontspec 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.
<code>\fontspec_set_family:Nnn</code>	<p>#1 : family #2 : fontspec features #3 : font name</p> <p>Defines a new font family from given <i>features</i> and <i>font</i>, and stores the name in the variable <i>family</i>. See the standard fontspec user commands for applications of this function.</p>

Part VI

The patching/improvement of \LaTeX 2_ε and other packages

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

19 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.

20 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 package.

21 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. This problem is fixed by

using the correct Unicode glyph, and the following packages are patched 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.

22 Discretionary hyphenation: \-

L^AT_EX defines the macro \- to insert discretionary hyphenation points. However, it is hard-coded in L^AT_EX 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 âĤ and now it does.

Part VII

fontspec.sty

23 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 \input binhex.tex % before expl syntax!
3 \ExplSyntaxOn

4 \msg_new:nnn {fontspec} {not-pdfTeX}
5 {
6   Requires~ XeTeX~ or~ LuaTeX~ to~ function!
7 }
8 \xetex_if_engine:F {
9   \luatex_if_engine:TF {
10     \RequirePackage{luatextra}[2010/05/10]
11     \luatexRequireModule{fontspec}
12   }{
13     \msg_error:nn {fontspec} {not-pdfTeX}
14   }
15 }

\xetex_or luatex:nn Use #1 if XeTeX or #2 if LuaTeX.
16 \xetex_if_engine:TF
17 { \cs_new_eq:NN \xetex_or luatex:nn \use_i:nn }
18 { \luatex_if_engine:T
19   { \cs_new_eq:NN \xetex_or luatex:nn \use_ii:nn }
20 }

\xetex_or luatex:nnn Use #1 and ({#2} if XeTeX) or ({#3} if LuaTeX).
21 \xetex_if_engine:TF
22 { \cs_new:Npn \xetex_or luatex:nnn #1#2#3 {#1{#2}} }
23 {
24   \luatex_if_engine:T
25   { \cs_new:Npn \xetex_or luatex:nnn #1#2#3 {#1{#3}} }
26 }
```

23.1 Bits and pieces

Conditionals

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

```

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

```

For dealing with legacy maths

```

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

```

For package options:

```

39 \newif\ifzf@configfile
40 \newif\ifzf@math

```

Counters

```

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

```

Temporary definition until expl3 has been updated to include this:

```

46 \cs_set:Npn \use:x #1 { \edef\@tempa{#1}\@tempa }
47 \cs_set:Npn \use_v:nnnnn #1#2#3#4#5 {#5}
48 \cs_set:Npn \use_iv:nnnnn #1#2#3#4#5 {#4}

```

\fontspec_glyph_if_exist:NnTF

```

49 \prg_new_conditional:Nnn \fontspec_glyph_if_exist:Nn {p,TF,T,F} {
50   \etex_ifontchar:D #1 #2 \scan_stop:
51   \prg_return_true:
52   \else:
53   \prg_return_false:
54   \fi:
55 }

56 \cs_new:Npn \fontspec_setkeys:xx #1#2
57 {
58   \use:x { \exp_not:N \setkeys*[zf]{#1}{#2} }
59 }
60 \cs_new:Npn \fontspec_setkeys:xxx #1#2#3
61 {
62   \use:x { \exp_not:N \setkeys*[zf@#1]{#2}{#3} }
63 }

```

23.2 Error/warning messages

Shorthands for messages:

```

64 \cs_new:Npn \fontspec_error:n { \msg_error:nn {fontspec} }
65 \cs_new:Npn \fontspec_error:nx { \msg_error:nnx {fontspec} }
66 \cs_new:Npn \fontspec_warning:n { \msg_warning:nn {fontspec} }

```

```

67 \cs_new:Npn \fontspec_warning:nx { \msg_warning:nnx {fontspec} }
68 \cs_new:Npn \fontspec_warning:nxx { \msg_warning:nnxx {fontspec} }
69 \cs_new:Npn \fontspec_info:n { \msg_info:nn {fontspec} }
70 \cs_new:Npn \fontspec_info:nx { \msg_info:nnx {fontspec} }
71 \cs_new:Npn \fontspec_info:nxx { \msg_info:nnxx {fontspec} }
72 \cs_new:Npn \fontspec_trace:n { \msg_trace:nn {fontspec} }

```

Errors:

```

73 \msg_new:nnn {fontspec} {no-size-info}
74 {
75   Size~ information~ must~ be~ supplied.\\
76   For~ example,~ SizeFeatures={Size={8-12},...}.
77 }
78 \msg_new:nnnn {fontspec} {rename-feature-not-exist}
79 {
80   The~ feature~ #1~ doesn't~ appear~ to~ be~ defined.
81 }
82 {
83   It~ looks~ like~ you're~ trying~ to~ rename~ a~ feature~ that~ doesn't~ exist.
84 }
85 \msg_new:nnn {fontspec} {no-glyph}
86 {
87   '\zf@fontname'~ does~ not~ contain~ glyph~ #1.
88 }
89 \msg_new:nnnn {fontspec} {unknown-options}
90 {
91   The~ following~ font~ options~ are~ not~ recognised:\\
92   \space\space\space\space #1
93 }
94 {
95   There~ is~ probably~ a~ typo~ in~ the~ font~ feature~ selection.
96 }
97 \msg_new:nnnn {fontspec} {euler-too-late}
98 {
99   The~ euler~ package~ must~ be~ loaded~ BEFORE~ fontspec.
100 }
101 {
102   fontspec~ only~ overwrites~ euler's~ attempt~ to\\
103   define~ the~ maths~ text~ fonts~ if~ fontspec~ is\\
104   loaded~ after~ euler.~ Type~ return~ to~ proceed\\
105   with~ incorrect~ \string\mathit,~ \string\mathbf,~ etc.
106 }
107 \msg_new:nnnn {fontspec} {no-xcolor}
108 {
109   Cannot~ load~ named~ colours~ without~ the~ xcolor~ package.
110 }
111 {
112   Sorry,~ I~ can't~ do~ anything~ to~ help.~ Instead~ of~ loading\\
113   the~ color~ package,~ use~ xcolor~ instead.~ It's~ better.
114 }
115 \msg_new:nnnn {fontspec} {unknown-color-model}
116 {

```



```

117 Error~ loading~ colour~ '#1';~ unknown~ colour~ model.
118 }
119 {
120 Sorry,~ I~ can't~ do~ anything~ to~ help.~ Please~ report~ this~ error\\
121 to~ my~ developer~ with~ a~ minimal~ example~ that~ causes~ the~ problem.
122 }

Warnings:

123 \msg_new:nnn {fontspec} {addfontfeatures-ignored}
124 {
125 \string\addfontfeature (s)~ ignored;\\
126 it~ cannot~ be~ used~ with~ a~ font~ that~ wasn't~ selected~ by~ fontspec.
127 }
128 \msg_new:nnn {fontspec} {feature-option-overwrite}
129 {
130 Option~ '#2'~ of~ font~ feature~ '#1'~ overwritten.
131 }
132 \msg_new:nnn {fontspec} {script-not-exist}
133 {
134 Font~ '\zf@fontname'~ does~ not~ contain~ script~ '#1'.
135 }
136 \msg_new:nnn {fontspec} {aat-feature-not-exist}
137 {
138 '\XKV@tfam=\XKV@tkey'~ feature~ not~ supported\\
139 for~ AAT~ font~ '\zf@fontname'.
140 }
141 \msg_new:nnn {fontspec} {aat-feature-not-exist-in-font}
142 {
143 AAT~ feature~ '\XKV@tfam=\XKV@tkey'~ (#1)~ not~ available\\
144 in~ font~ '\zf@fontname'.
145 }
146 \msg_new:nnn {fontspec} {icu-feature-not-exist}
147 {
148 '\XKV@tfam=\XKV@tkey'~ feature~ not~ supported\\
149 for~ ICU~ font~ '\zf@fontname'
150 }
151 \msg_new:nnn {fontspec} {icu-feature-not-exist-in-font}
152 {
153 OpenType~ feature~ '\XKV@tfam=\XKV@tkey'~ (#1)~ not~ available\\
154 for~ font~ '\zf@fontname', \\
155 with~ script~ '\l_fontspec_script_name_tl',~ and~ language~ '\l_fontspec_lang_name_tl'.
156 }
157 \msg_new:nnn {fontspec} {no-opticals}
158 {
159 '\zf@fontname'~ doesn't~ appear~ to~ have~ an~ Optical~ Size~ axis.
160 }
161 \msg_new:nnn {fontspec} {language-not-exist}
162 {
163 Language~ '#1'~ not~ available\\
164 for~ font~ '\zf@fontname'\\
165 with~ script~ '\l_fontspec_script_name_tl'.
166 }

```

```

167 \msg_new:nnn {fontspec} {only-xetex-feature}
168 {
169   Ignored~ XeTeX~ only~ feature:~ '#1'.
170 }
171 \msg_new:nnn {fontspec} {only-luatex-feature}
172 {
173   Ignored~ LuaTeX~ only~ feature:~ '#1'.
174 }
175 \msg_new:nnn {fontspec} {no-mapping}
176 {
177   Input~ mapping~ not~ (yet?)~ supported~ in~ LuaTeX.
178 }
179 \msg_new:nnn {fontspec} {no-mapping-ligtext}
180 {
181   Input~ mapping~ not~ (yet?)~ supported~ in~ LuaTeX.\
182   Use~ "Ligatures=TeX"~ instead~ of~ "Mapping=tex-text".
183 }
184 \msg_new:nnn {fontspec} {cm-default-obsolete}
185 {
186   The~ "cm-default"~ package~ option~ is~ obsolete.
187 }

```

Info messages:

```

188 \msg_new:nnn {fontspec} {defining-font}
189 {
190   Defining~ font~ family~ for~ '#2'~ with~ options~ [\zf@default@options #1].
191 }
192 \msg_new:nnn {fontspec} {no-font-shape}
193 {
194   Could~ not~ resolve~ font~ #1~ (it~ probably~ doesn't~ exist).
195 }
196 \msg_new:nnn {fontspec} {set-scale}
197 {
198   \zf@fontname\space scale ~~= \l_fontspec_scale_tl.
199 }
200 \msg_new:nnn {fontspec} {setup-math}
201 {
202   Adjusting~ the~ maths~ setup~ (use~ [no-math]~ to~ avoid~ this).
203 }
204 \msg_new:nnn {fontspec} {no-scripts}
205 {
206   Font~ \zf@fontname\space does~ not~ contain~ any~ OpenType~ 'Script'~ information.
207 }

```

23.3 Option processing

```

208 \DeclareOption{cm-default}{
209   \fontspec_warning:n {cm-default-obsolete}
210 }
211 \DeclareOption{math}{\@zf@mathtrue}
212 \DeclareOption{no-math}{\@zf@mathfalse}
213 \DeclareOption{config}{\@zf@configfiletrue}

```

```

214 \DeclareOption{no-config}{\@zf@configfilefalse}
215 \DeclareOption{quiet}{
216   \msg_redirect_module:nnn { fontspec } { warning } { info }
217   \msg_redirect_module:nnn { fontspec } { info } { none }
218 }
219 \DeclareOption{silent}{
220   \msg_redirect_module:nnn { fontspec } { warning } { none }
221   \msg_redirect_module:nnn { fontspec } { info } { none }
222 }
223 \ExecuteOptions{config,math}
224 \ProcessOptions*

```

23.4 Packages

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

```

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

```

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

```

227 \RequirePackage{fontspec-patches}

```

23.5 Encodings

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

```

228 \xetex_or luatex:nnn {\tl_set:Nn \zf@enc} {EU1} {EU2}
229 \tl_set:Nn \rmdefault {lmr}
230 \tl_set:Nn \sfdefault {lms}
231 \tl_set:Nn \ttdefault {lmtt}
232 \RequirePackage[\zf@enc]{fontenc}
233 \tl_set_eq:NN \UTFencname \zf@enc % for xunicode

```

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

```

234 \tl_set_eq:NN \cyrillicencoding \zf@enc
235 \tl_set_eq:NN \latinencoding \zf@enc
236 \g@addto@macro \document {
237   \tl_set_eq:NN \cyrillicencoding \zf@enc
238   \tl_set_eq:NN \latinencoding \zf@enc
239 }

```

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.

23.6 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 23.8 on page 60](#).

23.6.1 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.

```
240 \DeclareDocumentCommand \fontspec { O{} m } {
241   \fontspec_set_family:Nnn \f@family {#1}{#2}
242   \selectfont
243   \ignorespaces
244 }
```

`\setmainfont` 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.

```
245 \DeclareDocumentCommand \setmainfont { O{} m } {
246   \fontspec_set_family:Nnn \rmdefault {#1}{#2}
247   \normalfont
248 }
249 \DeclareDocumentCommand \setsansfont { O{} m } {
250   \fontspec_set_family:Nnn \sfdefault {#1}{#2}
251   \normalfont
252 }
253 \DeclareDocumentCommand \setmonofont { O{} m } {
254   \fontspec_set_family:Nnn \ttdefault {#1}{#2}
255   \normalfont
256 }
```

`\setromanfont` This is the old name for `\setmainfont`, retained for backwards compatibility.

```
257 \cs_set_eq:NN \setromanfont \setmainfont
```

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

`\setmathsf`

`\setboldmathrm` `\setboldmathrm` is used for specifying which fonts should be used in `\boldmath`.

```
\setmathtt 258 \DeclareDocumentCommand \setmathrm { O{} m } {
259   \fontspec_set_family:Nnn \zf@rmmaths {#1}{#2}
260 }
261 \DeclareDocumentCommand \setboldmathrm { O{} m } {
262   \fontspec_set_family:Nnn \zf@rmboldmaths {#1}{#2}
263 }
264 \DeclareDocumentCommand \setmathsf { O{} m } {
265   \fontspec_set_family:Nnn \zf@sfmaths {#1}{#2}
266 }
267 \DeclareDocumentCommand \setmathtt { O{} m } {
```

```

268 \fontspec_set_family:Nnn \zf@ttmaths {#1}{#2}
269 }
270 \@onlypreamble\setmathrm
271 \@onlypreamble\setboldmathrm
272 \@onlypreamble\setmathsf
273 \@onlypreamble\setmathtt

```

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

```

274 \def\zf@rmmaths{\rmdefault}
275 \def\zf@sfmaths{\sfdefault}
276 \def\zf@ttmaths{\ttdefault}

```

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

`\fontspec_select:nn` defines a font family and saves its name in `\zf@family`. This family is then used in a typical NFSS `\fontfamily` declaration, saved in the macro name specified.

```

277 \DeclareDocumentCommand \newfontfamily { m O{} m } {
278   \fontspec_select:nn{#2}{#3}
279   \use:x {
280     \exp_not:N \DeclareRobustCommand \exp_not:N #1 {
281       \exp_not:N \fontfamily {\zf@family} \exp_not:N \selectfont
282     }
283   }
284 }

```

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

```

285 \DeclareDocumentCommand \newfontface { m O{} m } {
286   \newfontfamily #1 [ BoldFont={},ItalicFont={},SmallCapsFont={},#2 ] {#3}
287 }

```

23.6.2 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.

```

288 \DeclareDocumentCommand \defaultfontfeatures {m} {\def\zf@default@options{#1,}}
289 \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.

```

290 \DeclareDocumentCommand \addfontfeatures {m} {
291   \ifcsname zf@family@fontdef\fontfamily\endcsname
292     \begingroup
293       \let\zf@default@options\@empty
294       \use:x {
295         \exp_not:N\fontspec_select:nn
296         {\csname zf@family@options\fontfamily\endcsname,#1}
297         {\csname zf@family@fontname\fontfamily\endcsname}
298       }
299     \endgroup
300     \fontfamily\zf@family\selectfont
301   \else
302     \fontspec_warning:n {addfontfeatures-ignored}
303   \fi
304   \ignorespaces
305 }
306 \let\addfontfeature\addfontfeatures

```

23.6.3 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.

```

307 \DeclareDocumentCommand \newfontfeature {mm} {
308   \stepcounter{zf@newff}
309   \cs_set:cpx{+zf-#1}{+zf-\the\c@zf@newff}
310   \define@key{zf}[options]{#1}[] {
311     \zf@update@family{\csname+zf-#1\endcsname}
312     \zf@update@ff{#2}
313   }
314 }

```

`\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.

```

315 \DeclareDocumentCommand \newAATfeature {mmm} {
316   \unless\ifcsname zf@options@#1\endcsname
317     \zf@define@font@feature{#1}
318   \fi
319   \key@ifundefined{zf}[#1]{#2}{ } {
320     \fontspec_warning:nxx {feature-option-overwrite}{#1}{#2}

```

```

321 }
322 \zf@define@feature@option{#1}{#2}{#3}{#4}{}
323 }

```

`\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.

```

324 \DeclareDocumentCommand \newICUfeature {mmm} {
325   \unless\ifcsname zf@options@#1\endcsname
326   \zf@define@font@feature{#1}
327   \fi
328   \key@ifundefined[zf]{#1}{#2}{}{
329     \fontspec_warning:nxx {feature-option-overwrite}{#1}{#2}
330   }
331   \zf@define@feature@option{#1}{#2}{}{#3}
332 }
333 \cs_set_eq:NN \newopentypefeature \newICUfeature

```

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

```

334 \DeclareDocumentCommand \aliasfontfeature {mm} {\multi@alias@key{#1}{#2}}
335 \DeclareDocumentCommand \aliasfontfeatureoption {mmm} {
336   \keyval@alias@key[zf@feat]{#1}{#2}{#3}
337 }

```

`\newfontscript` Mostly used internally, but also possibly useful for users, to define new OpenType 'scripts', mapping logical names to OpenType script tags. Iterates through 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.

```

338 \DeclareDocumentCommand \newfontscript {mm}
339 {
340   \fontspec_new_script:nn {#1} {#2}
341   \fontspec_new_script:nn {#2} {#2}
342 }
343 \cs_new:Npn \fontspec_new_script:nn #1#2
344 {
345   \define@key[zf@feat]{Script}{#1}[]{}
346   \fontspec_check_script:nTF {#2} {
347     \zf@update@family{+script=#1}
348     \tl_set:Nn \l_fontspec_script_tl {#2}
349     \c@zf@script=\l_fontspec_strnum_int\relax
350   }{
351     \fontspec_warning:nx {script-not-exist} {#1}
352   }
353 }
354 }

```

`\newfontlanguage` Mostly used internally, but also possibly useful for users, to define new OpenType 'languages', mapping logical names to OpenType language tags. Iterates through

the languages in the selected font to check that it's a valid feature choice, and then prepends the (X_YT_EX) \font feature string with the appropriate language selection tag.

```

355 \DeclareDocumentCommand \newfontlanguage {mm}
356 {
357   \fontspec_new_lang:nn {#1} {#2}
358   \fontspec_new_lang:nn {#2} {#2}
359 }

360 \cs_new:Npn \fontspec_new_lang:nn #1#2
361 {
362   \define@key[zf@feat]{Lang}{#1}[]{
363     \fontspec_check_lang:nTF {#2} {
364       \zf@update@family{+lang=#1}
365       \tl_set:Nn \l_fontspec_lang_tl {#2}
366       \c@zf@language=\l_fontspec_strnum_int\relax
367     }{
368       \fontspec_warning:nx {language-not-exist} {#1}
369     }
370   }
371 }

```

\DeclareFontExtensions dfont would never be uppercase, right?

```

372 \DeclareDocumentCommand \DeclareFontExtensions {m}
373 {
374   \tl_set:Nx \l_fontspec_extensions_clist { \zap@space #1~\@empty }
375 }
376 \DeclareFontExtensions{.otf,.ttf,.OTF,.TTF,.ttc,.TTC,.dfont}

```

23.7 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 Test whether the currently selected font has been loaded by fontspec.
377 \prg_new_conditional:Nnn \fontspec_if_fontspec_font: {TF,T,F} {
378   \ifcsname zf@family@fontdef\@family\endcsname
379   \prg_return_true:
380   \else
381   \prg_return_false:
382   \fi
383 }

```


`\fontspec_if_aat_feature:nnTF` Conditional to test if the currently selected font contains the AAT feature (#1,#2).

```

384 \prg_new_conditional:Nnn \fontspec_if_aat_feature:nn {TF,T,F} {
385   \ifcsname zf@family@fontdef\f@family\endcsname
386   \font\zf@basefont="\use:c{zf@family@fontdef\f@family}"~at~\f@size pt
387   \ifzf@atsui
388     \fontspec_make_AAT_feature_string:nn{#1}{#2}
389     \ifx\@tempa\@empty
390       \prg_return_false:
391     \else
392       \prg_return_true:
393     \fi
394   \else
395     \prg_return_false:
396   \fi
397 \else
398   \prg_return_false:
399 \fi
400 }
```

`\fontspec_if_opentype:TF` Test whether the currently selected font is an OpenType font. Always true for LuaTeX fonts.

```

401 \prg_new_conditional:Nnn \fontspec_if_opentype: {TF,T,F} {
402   \ifcsname zf@family@fontdef\f@family\endcsname
403   \font\zf@basefont="\csname zf@family@fontdef\f@family\endcsname"~at~\f@size pt
404   \fontspec_set_font_type:
405   \ifzf@icu
406     \prg_return_true:
407   \else
408     \prg_return_false:
409   \fi
410 \else
411   \prg_return_false:
412 \fi
413 }
```

`\fontspec_if_feature:nTF` Test 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.

```

414 \prg_new_conditional:Nnn \fontspec_if_feature:n {TF,T,F} {
415   \ifcsname zf@family@fontdef\f@family\endcsname
416   \font\zf@basefont="\csname zf@family@fontdef\f@family\endcsname"~at~\f@size pt
417   \fontspec_set_font_type:
418   \ifzf@icu
419     \int_set:Nn \c@zf@script
420     {\use:c {g_fontspec_script_num_(\zf@family)_tl}}
421     \int_set:Nn \c@zf@language
422     {\use:c {g_fontspec_lang_num_(\zf@family)_tl}}
423     \tl_set:Nv \l_fontspec_script_tl {g_fontspec_script_(\zf@family)_tl}
424     \tl_set:Nv \l_fontspec_lang_tl {g_fontspec_lang_(\zf@family)_tl}
425     \fontspec_check_ot_feat:nTF {#1} \prg_return_true: \prg_return_false:
426   \else
```

```

427     \prg_return_false:
428   \fi
429 \else
430   \prg_return_false:
431 \fi
432 }

```

`\fontspec_if_feature:nnnTF` 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.: `\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.

```

433 \prg_new_conditional:Nnn \fontspec_if_feature:nnn {TF,T,F} {
434   \ifcsname zf@family@fontdef\f@family\endcsname
435     \font\zf@basefont="\csname zf@family@fontdef\f@family\endcsname"~at~\f@size pt
436     \fontspec_set_font_type:
437     \ifzf@icu
438       \fontspec_iv_str_to_num:n{#1} \c@zf@script = \l_fontspec_strnum_int \relax
439       \fontspec_iv_str_to_num:n{#2} \c@zf@language = \l_fontspec_strnum_int \relax
440       \fontspec_check_ot_feat:nTF {#3} \prg_return_true: \prg_return_false:
441     \else
442       \prg_return_false:
443     \fi
444   \else
445     \prg_return_false:
446   \fi
447 }

```

`\fontspec_if_script:nTF` Test 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.

```

448 \prg_new_conditional:Nnn \fontspec_if_script:n {TF,T,F} {
449   \ifcsname zf@family@fontdef\f@family\endcsname
450     \font\zf@basefont="\csname zf@family@fontdef\f@family\endcsname"~at~\f@size pt
451     \fontspec_set_font_type:
452     \ifzf@icu
453       \fontspec_check_script:nTF {#1} \prg_return_true: \prg_return_false:
454     \else
455       \prg_return_false:
456     \fi
457   \else
458     \prg_return_false:
459   \fi
460 }

```

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

```

461 \prg_new_conditional:Nnn \fontspec_if_language:n {TF,T,F} {
462   \ifcsname zf@family@fontdef\f@family\endcsname
463     \font\zf@basefont="\csname zf@family@fontdef\f@family\endcsname"~at~\f@size pt
464     \fontspec_set_font_type:

```

```

465 \ifzf@icu
466 \tl_set:Nv \l_fontspec_script_tl {g_fontspec_script_(\zf@family)_tl}
467 \int_set:Nn \c_zf@script
468 {\use:c {g_fontspec_script_num_(\zf@family)_tl}}
469 \fontspec_check_lang:nTF {#1} \prg_return_true: \prg_return_false:
470 \else
471 \prg_return_false:
472 \fi
473 \else
474 \prg_return_false:
475 \fi
476 }

```

`\fontspec_if_language:nnTF` Test whether the currently selected font contains the raw OpenType language tag #2 in script #1. E.g.: `\fontspec_if_language:nnTF {cyr1} {SRB} {True} {False}`. Returns false if the font is not loaded by fontspec or is not an OpenType font.

```

477 \prg_new_conditional:Nnn \fontspec_if_language:nn {TF,T,F} {
478 \ifcsname zf@family@fontdef\fontspec_if_language\endcsname
479 \font\zf@basefont="\csname zf@family@fontdef\fontspec_if_language\endcsname"~at~\f@size pt
480 \fontspec_set_font_type:
481 \ifzf@icu
482 \tl_set:Nn \l_fontspec_script_tl {#1}
483 \fontspec_iv_str_to_num:n{#1} \c_zf@script = \l_fontspec_strnum_int \relax
484 \fontspec_check_lang:nTF {#2} \prg_return_true: \prg_return_false:
485 \else
486 \prg_return_false:
487 \fi
488 \else
489 \prg_return_false:
490 \fi
491 }

```

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

```

492 \prg_new_conditional:Nnn \fontspec_if_current_script:n {TF,T,F} {
493 \ifcsname zf@family@fontdef\fontspec_if_current_script\endcsname
494 \font\zf@basefont="\csname zf@family@fontdef\fontspec_if_current_script\endcsname"~at~\f@size pt
495 \fontspec_set_font_type:
496 \ifzf@icu
497 \tl_if_eq:nvTF {#1} {g_fontspec_script_(\zf@family)_tl}
498 {\prg_return_true:} {\prg_return_false:}
499 \else
500 \prg_return_false:
501 \fi
502 \else
503 \prg_return_false:
504 \fi
505 }

```

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

```

506 \prg_new_conditional:Nnn \fontspec_if_current_language:n {TF,T,F} {
507   \ifcsname zf@family@fontdef\f@family\endcsname
508     \font\zf@basefont="\csname zf@family@fontdef\f@family\endcsname"~at~\f@size pt
509   \fontspec_set_font_type:
510   \ifzf@icu
511     \tl_if_eq:nvTF {#1} {g_fontspec_lang_(\zf@family)_tl}
512       {\prg_return_true:} {\prg_return_false:}
513   \else
514     \prg_return_false:
515   \fi
516 \else
517   \prg_return_false:
518 \fi
519 }

```

Need this:

```
520 \cs_generate_variant:Nn \tl_if_eq:nnTF {nv}
```

```

\fontspec_set_family:Nnn #1 : family
                        #2 : fontspec features
                        #3 : font name

```

Defines a new font family from given *⟨features⟩* and *⟨font⟩*, and stores the name in the variable *⟨family⟩*. See the standard fontspec user commands for applications of this function.

We want to store the actual name of the font family within the *⟨family⟩* variable because the actual L^AT_EX family name is automatically generated by fontspec and it's easier to keep it that way.

Please use `\fontspec_set_family:Nnn` instead of `\fontspec_select:nn`, which may change in the future.

```

521 \cs_new:Npn \fontspec_set_family:Nnn #1#2#3 {
522   \fontspec_select:nn {#2}{#3}
523   \tl_set_eq:NN #1 \zf@family
524 }

```

23.8 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 proce-
                    dure of all \fontspec-like commands. Given a list of font features (#1) for a re-
                    quested font (#2), it will define an NFSS family for that font and put the family
                    name (globally) into \zf@family. The TEX '\font' command is (globally) stored in
                    \zf@basefont.

```

This macro does its processing inside a group to attempt to restrict the scope of its internal processing. This works to some degree to insulate the internal commands from having to be manually.

```

525 \cs_set:Npn \fontspec_select:nn #1#2 {
526   \begingroup
527   \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.

```
528 \edef\zf@fontname{#2}
529 \let\zf@family@long\zf@fontname
530 \let\zf@up\zf@fontname
```

Now convert the requested features to font definition strings. First the features are parsed for information about font loading (whether it's a named font or external font, etc.), and then information is extracted for the names of the other shape fonts.

Then the mapping from user features to low-level features occurs. This is performed with \fontspec_get_features:n, in which \setkeys retrieves the requested font features and processes them. As \setkeys is run multiple times, some of its information storing only occurs once while we decide if the font family has been defined or not. When the later processing is occurring per-shape this no longer needs to happen; this is indicated by the 'firsttime' conditional.

```
531 \fontspec_preparse_features:nn {#1}{#2}
532 \fontspec_set_scriptlang:
533 \fontspec_get_features:n {\zf@font@feat}
534 \zf@firsttimefalse
```

Check if the family is unique and, if so, save its information. (\addfontfeature and other macros use this data.) Then 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.

```
535 \fontspec_save_family:nT {#2} {
536   \fontspec_info:nxx {defining-font} {#1} {#2}
537   \fontspec_save_fontinfo:nn {#1} {#2}
538   \DeclareFontFamily{\zf@enc}{\zf@family}{}
539   \fontspec_set_upright:
540   \fontspec_set_bold:
541   \fontspec_set_italic:
542   \fontspec_set_slanted:
543   \fontspec_set_bold_italic:
544   \fontspec_set_bold_slanted:
545 }
546 \endgroup
547 }
```

\zf@fontspec For backwards compatibility. Do not use this from now on!

```
548 \cs_set_eq:NN \zf@fontspec \fontspec_select:nn
```

\fontspec_preparse_features:nn Perform the (multi-step) feature parsing process.

```
549 \cs_new:Npn \fontspec_preparse_features:nn #1#2 {
```

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

```
550 \fontspec_if_detect_external:nT {#2}
```

```

551 { \setkeys[zf]{preparse-external}{ExternalLocation} }
552 \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.

```

553 \let\zf@fontname\zf@up
554 \fontspec_setkeys:xx {preparse} {\XKV@rm}
555 \let\zf@font@feat\XKV@rm

```

Finally save the ‘confirmed’ font definition.

```

556 \font\zf@basefont="\fontspec_fullname:n {\zf@up}"~at~\f@size pt
557 \fontspec_set_font_type:
558 \global\font\zf@basefont="\fontspec_fullname:n {\zf@up}"~at~\f@size pt
559 \zf@basefont % this is necessary for LuaLaTeX to check the scripts properly
560 }

```

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

```

561 \prg_new_conditional:Nnn \fontspec_if_detect_external:n {T}
562 {
563   \clist_map_inline:Nn \l_fontspec_extensions_clist
564   {
565     \bool_set_false:N \l_tmpa_bool
566     \tl_if_in:nnT {#1 = end_of_string} {##1 = end_of_string}
567     { \bool_set_true:N \l_tmpa_bool \clist_map_break: }
568   }
569   \bool_if:NTF \l_tmpa_bool \prg_return_true: \prg_return_false:
570 }

```

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

```

571 \cs_set:Npn \fontspec_fullname:n #1 {
572   \fontspec_namewrap:n { #1 \l_fontspec_extension_tl }
573   \l_fontspec_renderer_tl
574   \l_fontspec_optical_size_tl
575 }

```

`\fontspec_save_family:nT` 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.

```

576 \prg_new_conditional:Nnn \fontspec_save_family:n {T} {
577   \unless\ifcsname zf@UID@\zf@family@long\endcsname
578   \ifcsname c@zf@famc@#1\endcsname
579     \expandafter\global\expandafter\advance
580     \csname c@zf@famc@#1\endcsname \@ne
581   \else
582     \expandafter\global\expandafter\newcount
583     \csname c@zf@famc@#1\endcsname
584   \fi
585   \edef\@tempa{#1~}
586   \cs_gset:cpx{zf@UID@\zf@family@long}{
587     \expandafter\zap@space\@tempa\@empty

```

```

588     (\expandafter\the\csname c@zf@famc@#1\endcsname)
589   }
590 \fi
591 \xdef\zf@family{\@nameuse{zf@UID@\zf@family@long}}
592 \cs_if_exist:cTF {zf@family@fontname\zf@family}
593   \prg_return_false: \prg_return_true:
594 }

```

`\fontspec_set_scriptlang:` Only necessary for OpenType fonts. First check if the font supports scripts, then apply defaults if none are explicitly requested. Similarly with the language settings.

```

595 \cs_new:Npn \fontspec_set_scriptlang: {
596   \ifzf@icu
597     \tl_if_empty:NTF \l_fontspec_script_name_tl {
598       \fontspec_check_script:nTF {latn}
599       {
600         \tl_set:Nn \l_fontspec_script_name_tl {Latin}
601         \tl_if_empty:NT \l_fontspec_lang_name_tl {
602           \tl_set:Nn \l_fontspec_lang_name_tl {Default}
603         }
604         \fontspec_setkeys:xxx {feat} {Script} {\l_fontspec_script_name_tl}
605         \fontspec_setkeys:xxx {feat} {Lang}   {\l_fontspec_lang_name_tl}
606       }
607     {
608       \fontspec_info:n {no-scripts}
609     }
610   }
611   {
612     \tl_if_empty:NT \l_fontspec_lang_name_tl {
613       \tl_set:Nn \l_fontspec_lang_name_tl {Default}
614     }
615     \fontspec_setkeys:xxx {feat} {Script} {\l_fontspec_script_name_tl}
616     \fontspec_setkeys:xxx {feat} {Lang}   {\l_fontspec_lang_name_tl}
617   }
618 \fi
619 }

```

`\fontspec_save_fontinfo:nn` Saves the relevant font information for future processing.

```

620 \cs_new:Npn \fontspec_save_fontinfo:nn #1#2 {
621   \tl_gset:cx {zf@family@fontname\zf@family} {#2}
622   \tl_gset:cx {zf@family@options\zf@family} {\zf@default@options #1}
623   \tl_gset:cx {zf@family@fontdef\zf@family} {
624     \fontspec_fullname:n {\zf@fontname} : \l_fontspec_pre_feat_tl \l_fontspec_rawfeatures_sclist
625   }
626   \tl_gset:cx {g_fontspec_script_num_(\zf@family)_tl}
627     { \int_use:N \c@zf@script }
628   \tl_gset:cx {g_fontspec_lang_num_(\zf@family)_tl}
629     { \int_use:N \c@zf@language }
630   \tl_gset_eq:cN {g_fontspec_script_(\zf@family)_tl} \l_fontspec_script_tl
631   \tl_gset_eq:cN {g_fontspec_lang_(\zf@family)_tl}   \l_fontspec_lang_tl
632 }

```

`\fontspec_set_upright:` Sets the upright shape.

```
633 \cs_new:Npn \fontspec_set_upright: {  
634   \zf@make@font@shapes{\zf@fontname}  
635   {\mddefault}{\updefault}{\zf@font@feat\zf@up@feat}  
636 }
```

`\fontspec_set_bold:` 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.

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.

```
637 \cs_new:Npn \fontspec_set_bold: {  
638   \unless\ifzf@nobf  
639     \ifx\zf@bf\@empty  
640       \zf@make@font@shapes[\zf@fontname]{/B}  
641       {\bfdefault}{\updefault}{\zf@font@feat\zf@bf@feat}  
642     \else  
643       \zf@make@font@shapes{\zf@bf}  
644       {\bfdefault}{\updefault}{\zf@font@feat\zf@bf@feat}  
645     \fi  
646   \fi  
647 }
```

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

```
648 \cs_new:Npn \fontspec_set_italic: {  
649   \unless\ifzf@noit  
650     \ifx\zf@it\@empty  
651       \zf@make@font@shapes[\zf@fontname]{/I}  
652       {\mddefault}{\itdefault}{\zf@font@feat\zf@it@feat}  
653     \else  
654       \zf@make@font@shapes{\zf@it}  
655       {\mddefault}{\itdefault}{\zf@font@feat\zf@it@feat}  
656     \fi  
657   \fi  
658 }
```

`\fontspec_set_slanted:` And slanted but only if requested:

```
659 \cs_new:Npn \fontspec_set_slanted: {  
660   \ifx\zf@sl\@empty\else  
661     \zf@make@font@shapes{\zf@sl}  
662     {\mddefault}{\sldefault}{\zf@font@feat\zf@sl@feat}  
663   \fi  
664 }
```

`\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.

```
665 \cs_new:Npn \fontspec_set_bold_italic: {
```



```

666 \@tempswatruue
667 \ifzf@nobf\@tempswafalse\fi
668 \ifzf@noit\@tempswafalse\fi
669 \bool_if:NT \l_fontspec_external_bool \@tempswafalse
670 \if@tempswa
671   \ifx\zf@bfit\@empty
672     \ifx\zf@bf\@empty
673       \ifx\zf@it\@empty
674         \zf@make@font@shapes[\zf@fontname]{/BI}
675         {\bfdefault}{\itdefault}{\zf@font@feat\zf@bfit@feat}
676       \else
677         \zf@make@font@shapes[\zf@it]{/B}
678         {\bfdefault}{\itdefault}{\zf@font@feat\zf@bfit@feat}
679       \fi
680     \else
681       \zf@make@font@shapes[\zf@bf]{/I}
682       {\bfdefault}{\itdefault}{\zf@font@feat\zf@bfit@feat}
683     \fi
684   \else
685     \zf@make@font@shapes{\zf@bfit}
686     {\bfdefault}{\itdefault}{\zf@font@feat\zf@bfit@feat}
687   \fi
688 \fi
689 }

```

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

```

690 \cs_new:Npn \fontspec_set_bold_slanted: {
691   \ifx\zf@bfs\@empty
692     \ifx\zf@sl\@empty\else
693       \zf@make@font@shapes[\zf@sl]{/B}
694       {\bfdefault}{\sldefault}{\zf@font@feat\zf@bfs\@feat}
695     \fi
696   \else
697     \zf@make@font@shapes{\zf@bfs}
698     {\bfdefault}{\sldefault}{\zf@font@feat\zf@bfs\@feat}
699   \fi
700 }

```

23.8.1 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.

```

701 \xetex_or luatex:nnn { \cs_new:Npn \fontspec_set_font_type: }
702 {
703   \zf@tfmfalse \zf@atsuifalse \zf@icufalse \zf@mmfalse \zf@graphitfalse
704   \ifcase\XeTeXfonttype\zf@basefont
705     \zf@tfmtrue
706   \or

```

```

707     \zf@atsuitrue
708     \ifnum\XeTeXcountvariations\zf@basefont \c_zero
709     \zf@mmtrue
710     \fi
711 \or
712     \zf@icutrue
713 \fi

```

If automatic, the `\l_fontspec_renderer_tl` token list will still be empty (other suffixes 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.

```

714 \tl_if_empty:NT \l_fontspec_renderer_tl {
715     \ifzf@atsui
716         \tl_set:Nn \l_fontspec_renderer_tl {/AAT}
717     \else\ifzf@icu
718         \tl_set:Nn \l_fontspec_renderer_tl {/ICU}
719     \fi\fi
720 }
721 }
722 {
723     \zf@icutrue
724 }

```

`\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.

```

725 \newcommand*\zf@make@font@shapes[5][] {
726     \begingroup
727     \edef\@tempa{#1}
728     \unless\ifx\@tempa\@empty
729         \font\@tempfonta="\fontspec_fullname:n {#1}"~at~\f@size pt
730         \edef\@tempa{\fontname\@tempfonta}
731     \fi
732     \font\@tempfontb="\fontspec_fullname:n {#1#2}"~at~\f@size pt
733     \edef\@tempb{\fontname\@tempfontb}
734     \ifx\@tempa\@tempb
735         \fontspec_info:nx {no-font-shape} {#1#2}
736     \else
737         \edef\zf@fontname{#1#2}
738         \let\zf@basefont\@tempfontb
739         \fontspec_declare_shape:nnnn {}{#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 23.10 on page 94](#) for the code that enables this usage.

```

740     \ifx\zf@sc\@empty
741     \unless\ifzf@nosc
742     \zf@make@smallcaps
743     \unless\ifx\zf@smallcaps\@empty
744     \fontspec_declare_shape:nnnn {\zf@smallcaps}{#3}
745     {\ifx#4\itdefault\sidefault\else\scdefault\fi}{#5\zf@sc@feat}
746     \fi
747     \fi
748     \else
749     \edef\zf@fontname{\zf@sc}
750     \fontspec_declare_shape:nnnn {}{#3}
751     {\ifx#4\itdefault\sidefault\else\scdefault\fi}{#5\zf@sc@feat}
752     \fi
753     \fi
754 \endgroup
755 }
```

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

`\fontspec_declare_shape:nnnn` #1 : Raw appended font feature
#2 : Font series
#3 : Font shape
#4 : Font features
Wrapper for `\DeclareFontShape`.

```

756 \cs_new:Npn \fontspec_declare_shape:nnnn #1#2#3#4 {
757   \clist_if_empty:NTF \l_fontspec_sizefeat_clist
758   {
759     \fontspec_get_features:n{#4}
760     \tl_set:Nx \l_fontspec_nfss_tl {
761       - \l_fontspec_scale_tl "
762       \fontspec_fullname:n {\zf@fontname} :
763       \l_fontspec_pre_feat_tl \l_fontspec_rawfeatures_sclist #1 "
764     }
765   }
```

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}}.`

```

766 {
767   \tl_clear:N \l_fontspec_nfss_tl
768   \clist_map_inline:Nn \l_fontspec_sizefeat_clist {
769     \tl_clear:N \l_fontspec_size_tl
770     \tl_set_eq:NN \l_fontspec_sizedfont_tl \zf@fontname
771     \fontspec_setkeys:xx {sizing} { \expandafter \@firstofone ##1 }
```

```

772 \tl_if_empty:NT \l_fontspec_size_tl { \fontspec_error:n {no-size-info} }
773 \fontspec_get_features:n{#4,\XKV@rm}
774 \tl_put_right:Nx \l_fontspec_nfss_tl {
775   \l_fontspec_size_tl \l_fontspec_scale_tl
776   " \fontspec_fullname:n { \l_fontspec_sizedfont_tl }
777   : \l_fontspec_pre_feat_tl \l_fontspec_rawfeatures_sclist #1 "
778 }
779 }
780 }

```

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.

```

781 \PackageInfo{fontspec}{
782   Defining~ shape~ #2/#3~ with~ raw~ font~ features:
783   \MessageBreak \l_fontspec_rawfeatures_sclist
784   \@gobble}
785 \use:x{
786   \exp_not:N \DeclareFontShape{\zf@enc}{\zf@family}{#2}{#3}
787     {\l_fontspec_nfss_tl}{\zf@adjust}
788 }

```

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?

```

789 \tl_if_eq:xxT {#3} {\itdefault}
790 {
791   \use:x {
792     \exp_not:N \DeclareFontShape {\zf@enc}{\zf@family}{#2}{\sldefault}
793       {-ssub*\zf@family/#2/\itdefault}{\zf@adjust}
794   }
795 }
796 }

```

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

```

797 \xetex_or luatex:nnn { \tl_set:Nn \l_fontspec_pre_feat_tl }
798 {
799   \ifzf@icu
800     \tl_if_empty:NF \l_fontspec_script_tl
801     {
802       script = \l_fontspec_script_tl ;
803       language = \l_fontspec_lang_tl ;
804     }
805   \fi
806 }
807 {
808   mode = \l_fontspec_mode_tl ;
809   \tl_if_empty:NF \l_fontspec_script_tl
810   {
811     script = \l_fontspec_script_tl ;
812     language = \l_fontspec_lang_tl ;
813   }

```

```
814 }
```

`\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:n` only the first time `\f@get@feature@requests` is called, so that the family name is only created once.

```
815 \newcommand*{\zf@update@family}[1]{
816   \ifzf@firsttime
817     \xdef\zf@family@long{\zf@family@long#1}
818   \fi
819 }
```

23.8.2 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.

```
820 \cs_set:Npn \fontspec_get_features:n #1 {
821   \let\l_fontspec_rawfeatures_sclist \empty
822   \tl_clear:N \l_fontspec_scale_tl
823   \let\zf@adjust \empty
824   \fontspec_setkeys:xx {options} {#1}
825   \tl_if_empty:NF \XKV@rm {
826     \fontspec_error:nx {unknown-options} { \exp_not:V \XKV@rm }
827   }
```

Finish the colour specification:

```
828   \cs_if_exist:NT \l_fontspec_hexcol_tl {
829     \zf@update@ff{color=\l_fontspec_hexcol_tl\l_fontspec_opacity_tl}
830   }
831 }
```

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

```
832 \tl_clear:N \zf@bf
833 \tl_clear:N \zf@it
834 \tl_clear:N \zf@fake@slant
835 \tl_clear:N \zf@fake@bolden
836 \tl_clear:N \zf@bfit
837 \tl_clear:N \zf@sl
838 \tl_clear:N \zf@bfsl
839 \tl_clear:N \zf@sc
840 \tl_clear:N \zf@up@feat
841 \tl_clear:N \zf@bf@feat
842 \tl_clear:N \zf@it@feat
843 \tl_clear:N \zf@bfit@feat
844 \tl_clear:N \zf@sl@feat
845 \tl_clear:N \zf@bfsl@feat
846 \tl_clear:N \zf@sc@feat
847 \tl_clear:N \l_fontspec_script_name_tl
848 \tl_clear:N \l_fontspec_script_tl
849 \tl_clear:N \l_fontspec_lang_name_tl
```

```

850 \tl_clear:N \l_fontspec_lang_tl
851 \clist_clear:N \l_fontspec_sizefeat_clist
852 \tl_set:Nn \l_fontspec_opacity_tl {FF}

```

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

```

853 \newcommand*\fontspec_init:{
854   \zf@icufalse
855   \zf@firsttimetrue
856   \xetex_or luatex:nnn { \cs_set:Npn \fontspec_namewrap:n ##1 }
857     { ##1 }
858     { name:##1 }
859   \tl_clear:N \l_fontspec_optical_size_tl
860   \tl_clear:N \l_fontspec_renderer_tl
861   \luatex_if_engine:T {
862     \tl_set:Nn \l_fontspec_mode_tl {node}
863     \luatexprehyphenchar = '\- % fixme
864     \luatexposthyphenchar = 0 % fixme
865     \luatexpreexhyphenchar = 0 % fixme
866     \luatexpostexhyphenchar = 0 % fixme
867   }
868 }

```

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

```

869 \newcommand*\zf@make@smallcaps{
870   \let\zf@smallcaps\@empty
871   \xetex_or luatex:nn
872   {
873     \ifzf@atsui
874       \fontspec_make_AAT_feature_string:nn{3}{3}
875       \unless\ifx\@tempa\@empty
876         \edef\zf@smallcaps{\@tempa;}
877       \fi
878     \fi
879     \ifzf@icu
880       \fontspec_check_ot_feat:nT {+smcp} {\edef\zf@smallcaps{+smcp;}}
881     \fi
882   }
883   {
884     \fontspec_check_ot_feat:nT {+smcp} {\edef\zf@smallcaps{+smcp;}}
885   }
886 }

```

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

```

887 \cs_new:Npn \sclist_put_right:Nn #1#2 {
888   \tl_if_empty:NTF #1 {
889     \tl_set:Nn #1 {#2}
890   }{
891     \tl_put_right:Nn #1 {;#2}
892   }
893 }

```

`\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.

```

894 \newcommand*\zf@update@ff[1]{
895   \unless\ifzf@firsttime
896     \xdef\l_fontspec_rawfeatures_sclist{\l_fontspec_rawfeatures_sclist #1;}
897   \fi
898 }

```

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

```

899 \cs_new:Npn \fontspec_make_feature:nnn #1#2#3 {
900   \xetex_or luatex:nn
901   {
902     \ifzf@atsui
903       \fontspec_make_AAT_feature:nn {#1}{#2}
904     \fi
905     \ifzf@icu
906       \fontspec_make_ICU_feature:n {#3}
907     \fi
908   }
909   {
910     \fontspec_make_ICU_feature:n {#3}
911   }
912 }
913 \cs_generate_variant:Nn \fontspec_make_feature:nnn {nnx}
914 \cs_new:Npn \fontspec_make_AAT_feature:nn #1#2 {
915   \tl_if_empty:nTF {#1}
916   {
917     \fontspec_warning:n {aat-feature-not-exist}
918   }
919   {
920     \fontspec_make_AAT_feature_string:nn {#1}{#2}
921     \ifx\@tempa\@empty
922       \fontspec_warning:nx {aat-feature-not-exist-in-font} {#1,#2}
923     \else
924       \zf@update@family{+#1,#2}
925       \zf@update@ff\@tempa
926     \fi
927   }
928 }
929 \cs_new:Npn \fontspec_make_ICU_feature:n #1 {
930   \tl_if_empty:nTF {#1}
931   {
932     \fontspec_warning:n {icu-feature-not-exist}
933   }
934   {
935     \fontspec_check_ot_feat:nTF {#1} {
936       \zf@update@family{#1}
937       \zf@update@ff{#1}
938     }{

```

```

939     \fontspec_warning:nx {icu-feature-not-exist-in-font} {#1}
940   }
941 }
942 }

```

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

```

\zf@define@feature@option 943 \newcommand*\zf@define@font@feature[1]{
944   \define@key[zf]{options}{#1}{\setkeys[zf@feat]{#1}{##1}}
945 }
946 \newcommand*\zf@define@feature@option[5]{
947   \define@key[zf@feat]{#1}{#2}[\fontspec_make_feature:nnn{#3}{#4}{#5}]
948 }

```

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

```

949 \newcommand*\keyval@alias@key[4][KV]{
950   \cs_set_eq:cc{#1@#2@#4}{#1@#2@#3}
951   \cs_set_eq:cc{#1@#2@#4@default}{#1@#2@#3@default}
952 }

```

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

```

953 \newcommand*\multi@alias@key[2]{
954   \key@ifundefined[zf]{options}{#1}
955   {
956     \key@ifundefined[zf]{preparse}{#1}
957     {
958       \key@ifundefined[zf]{preparse-external}{#1}
959       { \fontspec_warning:nx {rename-feature-not-exist} {#1} }
960       { \keyval@alias@key[zf]{preparse-external}{#1}{#2} }
961     }
962     { \keyval@alias@key[zf]{preparse}{#1}{#2} }
963   }
964   { \keyval@alias@key[zf]{options}{#1}{#2} }
965 }

```

`\fontspec_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 70).

```

966 \cs_new:Npn \fontspec_make_AAT_feature_string:nn #1#2 {
967   \edef\@tempa{\XeTeXfeaturename\zf@basefont #1}
968   \unless\ifx\@tempa\@empty

```

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

```

969     \ifnum\XeTeXisexclusivefeature\zf@basefont #10
970     \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:

```

971     \else
972     \unless\ifodd #2
973     \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.

```

974     \else
975         \edef\@tempb{
976             \XeTeXselectorname\zf@basefont #1\space \numexpr#2-1\relax
977         }
978         \unless\ifx\@tempb\@empty
979             \edef\@tempb{!\@tempb}
980         \fi
981     \fi
982 \fi

```

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

```

983     \unless\ifx\@tempb\@empty
984         \edef\@tempa{\@tempa=\@tempb}
985     \else
986         \let\@tempa\@empty
987     \fi
988 \fi
989 }

```

\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 \@empty and anything beyond four chars is snipped. The \@empty 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.

```

990 \cs_set:Npn \fontspec_iv_str_to_num:n #1 {
991     \fontspec_iv_str_to_num:w #1 \@empty \@empty \q_nil
992 }
993 \cs_set:Npn \fontspec_iv_str_to_num:w #1#2#3#4#5 \q_nil {
994     \int_set:Nn \l_fontspec_strnum_int {
995         '#1 * "1000000
996         + '#2 * "10000
997         + \ifx \@empty #3 32 \else '#3 \fi * "100
998         + \ifx \@empty #4 32 \else '#4 \fi
999     }
1000 }

```

```

1001 \cs_set:Npn \fontspec_v_str_to_num:n #1 {
1002   \bool_if:nTF
1003     {
1004       \tl_if_head_eq_charcode_p:nN {#1} {+} ||
1005       \tl_if_head_eq_charcode_p:nN {#1} {-}
1006     }
1007     {
1008       \exp_after:wN \fontspec_iv_str_to_num:n
1009       \exp_after:wN { \use_none:n #1 }
1010     }
1011     { \fontspec_iv_str_to_num:n {#1} }
1012 }

```

`\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.

```

1013 \xetex_or luatex:nnn {\prg_new_conditional:Nnn \fontspec_check_script:n {TF}}
1014 {
1015   \fontspec_iv_str_to_num:n{#1}
1016   \@tempcntb\XeTeXOTcountscripts\zf@basefont
1017   \c@zf@index\z@ \@tempwafalse
1018   \loop\ifnum\c@zf@index\@tempcntb
1019     \ifnum\XeTeXOTscripttag\zf@basefont\c@zf@index=\l_fontspec_strnum_int
1020       \@tempwattrue
1021       \c@zf@index\@tempcntb
1022     \else
1023       \advance\c@zf@index\@ne
1024     \fi
1025   \repeat
1026   \if@tempswa \prg_return_true: \else: \prg_return_false: \fi:
1027 }
1028 {
1029   \directlua{fontspec.check_ot_script("zf@basefont", "#1")}
1030   \if@tempswa \prg_return_true: \else: \prg_return_false: \fi:
1031 }

```

`\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'.

```

1032 \xetex_or luatex:nnn {\prg_new_conditional:Nnn \fontspec_check_lang:n {TF}}
1033 {
1034   \fontspec_iv_str_to_num:n{#1}
1035   \@tempcntb\XeTeXOTcountlanguages\zf@basefont\c@zf@script
1036   \c@zf@index\z@
1037   \@tempwafalse
1038   \loop\ifnum\c@zf@index\@tempcntb
1039     \ifnum\XeTeXOTlanguagetag\zf@basefont\c@zf@script\c@zf@index=\l_fontspec_strnum_int
1040       \@tempwattrue
1041     \c@zf@index\@tempcntb

```

```

1042     \else
1043       \advance\c@zf@index\@ne
1044     \fi
1045   \repeat
1046   \if@tempswa \prg_return_true: \else: \prg_return_false: \fi:
1047 }
1048 {
1049   \directlua{
1050     fontspec.check_ot_lang( "zf@basefont", "#1", "\l_fontspec_script_tl" )
1051   }
1052   \if@tempswa \prg_return_true: \else: \prg_return_false: \fi:
1053 }

```

`\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'.

```

1054 \xetex_or luatex:nnn
1055 { \prg_new_conditional:Nnn \fontspec_check_ot_feat:n {TF,T} }
1056 {
1057   \@tempcntb\XeTeXOTcountfeatures\zf@basefont\c@zf@script\c@zf@language
1058   \fontspec_v_str_to_num:n {#1}
1059   \c@zf@index\z@
1060   \@tempswafalse
1061   \loop\ifnum\c@zf@index\@tempcntb
1062     \ifnum\XeTeXOTfeaturetag\zf@basefont\c@zf@script\c@zf@language
1063       \c@zf@index=\l_fontspec_strnum_int
1064       \@tempswatrue
1065       \c@zf@index\@tempcntb
1066     \else
1067       \advance\c@zf@index\@ne
1068     \fi
1069   \repeat
1070   \if@tempswa \prg_return_true: \else: \prg_return_false: \fi:
1071 }
1072 {
1073   \directlua{
1074     fontspec.check_ot_feat(
1075       "zf@basefont", "#1",
1076       "\l_fontspec_lang_tl", "\l_fontspec_script_tl"
1077     )
1078   }
1079   \if@tempswa \prg_return_true: \else: \prg_return_false: \fi:
1080 }

```

23.9 keyval definitions

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

23.9.1 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 \TeX tree. Otherwise, the argument given defines the file path of the font.

```
1081 \bool_new:N \l_fontspeg_external_bool
1082 \define@key[zf]{preparse-external}{ExternalLocation}[]{
1083   \zf@nobftrue
1084   \zf@noittrue
1085   \bool_set_true:N \l_fontspeg_external_bool
1086   \xetex_or luatex:nnn { \cs_gset:Npn \fontspec_namewrap:n ##1 }
1087     { [ #1 ##1 ] }
1088     { file: #1 ##1 }
1089   \xetex_if_engine:T { \setkeys[zf]{preparse}{Renderer=ICU} }
1090 }
1091 \aliasfontfeature{ExternalLocation}{Path}
```

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

```
1092 \define@key[zf]{preparse-external}{Extension}{
1093   \tl_set:Nn \l_fontspeg_extension_tl {#1}
1094   \bool_if:NF \l_fontspeg_external_bool {
1095     \setkeys*[zf]{preparse-external}{ExternalLocation}
1096   }
1097 }
1098 \tl_clear:N \l_fontspeg_extension_tl
```

23.9.2 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.

```
1099 \define@choicekey[zf]{preparse}{Renderer}[\l_tmpa_tl\l_tmpa_num]
1100   {AAT,ICU,Graphite,Full,Basic}{
1101     \zf@update@family{+rend:#1}
1102     \intexpr_compare:nTF {\l_tmpa_num 3} {
1103       \xetex_or luatex:nn
1104       {
1105         \tl_set:Nv \l_fontspeg_renderer_tl {g_fontspeg_renderer_tag_\l_tmpa_tl}
1106       }
1107     }
```

```

1108     \fontspec_warning:nx {only-xetex-feature} {Renderer=AAT/ICU/Graphite}
1109   }
1110 }{
1111   \xetex_or luatex:nn
1112   { \fontspec_warning:nx {only-luatex-feature} {Renderer=Full/Basic} }
1113   { \tl_set:Nv \l_fontspec_mode_tl {g_fontspec_mode_tag_\l_tmpa_tl} }
1114 }
1115 }
1116 \tl_set:cn {g_fontspec_renderer_tag_AAT} {/AAT}
1117 \tl_set:cn {g_fontspec_renderer_tag_ICU} {/ICU}
1118 \tl_set:cn {g_fontspec_renderer_tag_Graphite} {/GR}
1119 \tl_set:cn {g_fontspec_mode_tag_Full} {node}
1120 \tl_set:cn {g_fontspec_mode_tag_Basic} {base}

```

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

```

1121 \define@key[zf]{preparse}{Script}{
1122   \xetex_if_engine:T { \setkeys[zf]{preparse}{Renderer=ICU} }
1123   \tl_set:Nn \l_fontspec_script_name_tl {#1}
1124   \zf@update@family{+script:#1}
1125 }

```

Exactly the same:

```

1126 \define@key[zf]{preparse}{Language}{
1127   \xetex_if_engine:T { \setkeys[zf]{preparse}{Renderer=ICU} }
1128   \tl_set:Nn \l_fontspec_lang_name_tl {#1}
1129   \zf@update@family{+language:#1}
1130 }

```

23.9.3 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:

```

1131 \define@key[zf]{preparse-external}{UprightFont}{
1132   \fontspec_complete_fontname:Nn \zf@up {#1}
1133   \zf@update@family{up:#1}
1134 }

```

Bold:

```

1135 \define@key[zf]{preparse-external}{BoldFont}{
1136   \edef\@tempa{#1}
1137   \ifx\@tempa\empty
1138     \zf@nobftrue
1139     \zf@update@family{nobf}
1140   \else
1141     \zf@nobffalse
1142     \fontspec_complete_fontname:Nn \zf@bf {#1}
1143     \zf@update@family{bf:#1}
1144   \fi

```

1145 }

Same for italic:

```
1146 \define@key[zf]{preparse-external}{ItalicFont}{
1147   \edef\@tempa{#1}
1148   \ifx\@tempa\@empty
1149     \zf@noittrue
1150     \zf@update@family{noit}
1151   \else
1152     \zf@noitfalse
1153     \fontspec_complete_fontname:Nn \zf@it {#1}
1154     \zf@update@family{it:#1}
1155   \fi
1156 }
```

Simpler for bold+italic & slanted:

```
1157 \define@key[zf]{preparse-external}{BoldItalicFont}{
1158   \fontspec_complete_fontname:Nn \zf@bfit {#1}
1159   \zf@update@family{bfit:#1}
1160 }
1161 \define@key[zf]{preparse-external}{SlantedFont}{
1162   \fontspec_complete_fontname:Nn \zf@sl {#1}
1163   \zf@update@family{sl:#1}
1164 }
1165 \define@key[zf]{preparse-external}{BoldSlantedFont}{
1166   \fontspec_complete_fontname:Nn \zf@bfs1 {#1}
1167   \zf@update@family{bfs1:#1}
1168 }
```

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

```
1169 \define@key[zf]{options}{SmallCapsFont}{
1170   \edef\@tempa{#1}
1171   \ifx\@tempa\@empty
1172     \zf@nosctrue
1173     \zf@update@family{nosc}
1174   \else
1175     \zf@noscfalse
1176     \fontspec_complete_fontname:Nn \zf@esc {#1}
1177     \zf@update@family{sc:\zap@space #1~\@empty}
1178   \fi
1179 }
```

\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").

```
1180 \cs_set:Npn \fontspec_complete_fontname:Nn #1#2 {
1181   \tl_set:Nn #1 {#2}
1182   \tl_replace_all_in:Nnx #1 {*} {\zf@fontname}
1183 }
1184 \cs_generate_variant:Nn \tl_replace_all_in:Nnn {Nnx}
```

Features

```

1185 \define@key[zf]{preparse}{UprightFeatures}{
1186   \def\zf@up@feat{, #1}
1187   \zf@update@family{rmfeat:#1}
1188 }
1189 \define@key[zf]{preparse}{BoldFeatures}{
1190   \def\zf@bf@feat{, #1}
1191   \zf@update@family{bfeat:#1}
1192 }
1193 \define@key[zf]{preparse}{ItalicFeatures}{
1194   \def\zf@it@feat{, #1}
1195   \zf@update@family{itfeat:#1}
1196 }
1197 \define@key[zf]{preparse}{BoldItalicFeatures}{
1198   \def\zf@bfit@feat{, #1}
1199   \zf@update@family{bfitfeat:#1}
1200 }
1201 \define@key[zf]{preparse}{SlantedFeatures}{
1202   \def\zf@sl@feat{, #1}
1203   \zf@update@family{slfeat:#1}
1204 }
1205 \define@key[zf]{preparse}{BoldSlantedFeatures}{
1206   \def\zf@bfs@feat{, #1}
1207   \zf@update@family{bfsfeat:#1}
1208 }

```

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

```

1209 \define@key[zf]{options}{SmallCapsFeatures}{
1210   \unless\ifzf@firsttime\def\zf@sc@feat{, #1}\fi
1211   \zf@update@family{scfeat:\zap@space #1~\@empty}
1212 }

```

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

```

1213 \define@key[zf]{preparse}{SizeFeatures}{
1214   \tl_set:Nn \l_fontspeg_sizefeat_clist {#1}
1215   \zf@update@family{sizefeat:\zap@space #1~\@empty}
1216 }
1217 \define@key[zf]{sizing}{Size}{ \tl_set:Nn \l_fontspeg_size_tl {#1} }
1218 \define@key[zf]{sizing}{Font}{
1219   \fontspec_complete_fontname:Nn \l_fontspeg_sizedfont_tl {#1}
1220 }

```

23.9.4 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.

```

1221 \define@key[zf]{options}{Scale}{
1222   \prg_case_str:nnn {#1}
1223   {

```

```

1224     {MatchLowercase} { \fontspec_calc_scale:n {5} }
1225     {MatchUppercase} { \fontspec_calc_scale:n {8} }
1226   }
1227   { \tl_set:Nx \l_fontspec_scale_tl {#1} }
1228   \zf@update@family{+scale:\l_fontspec_scale_tl}
1229   \tl_set:Nx \l_fontspec_scale_tl { s*[\l_fontspec_scale_tl] }
1230 }

```

`\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_YTeX).

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.

```

1231 \cs_new:Npn \fontspec_calc_scale:n #1 {
1232   \group_begin:
1233     \rmfamily
1234     \fontspec_set_font_dimen:NnN \@tempdima {#1} \font
1235     \fontspec_set_font_dimen:NnN \@tempdimb {#1} \zf@basefont
1236     \dim_set:Nn \@tempdimc { 1pt*\@tempdima/\@tempdimb }
1237     \tl_gset:Nx \l_fontspec_scale_tl {\strip@pt\@tempdimc}
1238     \fontspec_info:n {set-scale}
1239   \group_end:
1240 }

```

`\fontspec_set_font_dimen:NnN` This function sets the dimension #1 (for font #3) to ‘fontdimen’ #2 for either font dimension 5 (x-height) or 8 (cap-height). If, for some reason, these return an incorrect ‘zero’ value (as `\fontdimen8` might for a . t_{fm} font), then we cheat and measure the height of a glyph. We assume in this case that the font contains either an ‘X’ or an ‘x’.

```

1241 \cs_new:Npn \fontspec_set_font_dimen:NnN #1#2#3
1242 {
1243   \dim_set:Nn #1 { \fontdimen #2 #3 }
1244   \dim_compare:nNnT #1 = {0pt} {
1245     \settoheight #1 {
1246       \tl_if_eq:nnTF {#3} {\font} \rmfamily #3
1247       \prg_case_int:nnn #2 {
1248         {5} {x} % x-height
1249         {8} {X} % cap-height
1250       } {?} % "else" clause; never reached.
1251     }
1252   }
1253 }

```

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

```

1254 \define@key[zf]{options}{WordSpace}{
1255   \zf@update@family{+wordspace:#1}
1256   \unless\ifzf@firsttime

```



```

1257 \zf@wordspace@parse#1,\zf@ii,\zf@iii,\zf@
1258 \fi
1259 }

```

`\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}`.

```

1260 \def\zf@wordspace@parse#1,#2,#3,#4\zf@{
1261 \def\@tempa{#4}
1262 \ifx\@tempa\empty
1263 \setlength\@tempdima{#1\fontdimen2\zf@basefont}
1264 \@tempdimb\@tempdima
1265 \@tempdimc\@tempdima
1266 \else
1267 \setlength\@tempdima{#1\fontdimen2\zf@basefont}
1268 \setlength\@tempdimb{#2\fontdimen3\zf@basefont}
1269 \setlength\@tempdimc{#3\fontdimen4\zf@basefont}
1270 \fi
1271 \edef\zf@adjust{
1272 \zf@adjust
1273 \fontdimen2\font\the\@tempdima
1274 \fontdimen3\font\the\@tempdimb
1275 \fontdimen4\font\the\@tempdimc
1276 }
1277 }

```

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

```

1278 \define@key[zf]{options}{PunctuationSpace}{
1279 \zf@update@family{+punctspace:#1}
1280 \setlength\@tempdima{#1\fontdimen7\zf@basefont}
1281 \edef\zf@adjust{\zf@adjust\fontdimen7\font\the\@tempdima}
1282 }

```

Letterspacing

```

1283 \define@key[zf]{options}{LetterSpace}{
1284 \zf@update@family{+tracking:#1}
1285 \zf@update@ff{letterspace=#1}
1286 }

```

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.

```

1287 \define@key[zf]{options}{HyphenChar}{
1288 \zf@update@family{+hyphenchar:#1}
1289 \tl_if_eq:nnTF {#1} {None}
1290 {
1291 \tl_put_right:Nn \zf@adjust { \hyphenchar \font = \c_minus_one }
1292 }
1293 {
1294 \tl_if_single:nTF {#1}

```

```

1295     { \tl_set:Nn \l_fontspec_hyphenchar_tl {'#1} }
1296     { \tl_set:Nn \l_fontspec_hyphenchar_tl { #1} }
1297 \fontspec_glyph_if_exist:NnTF \zf@basefont {\l_fontspec_hyphenchar_tl}
1298 {
1299     \xetex_or luatex:nnn { \tl_put_right:Nn \zf@adjust }
1300     { \hyphenchar \font = \l_fontspec_hyphenchar_tl \scan_stop: }
1301     {
1302         \hyphenchar \font = \c_zero
1303         \luatexprehyphenchar = \l_fontspec_hyphenchar_tl \scan_stop:
1304     }
1305 }
1306 { \fontspec_error:nx {no-glyph}{#1} }
1307 }
1308 }

```

Color

```

1309 \define@key[zf]{options}{Color}{
1310     \zf@update@family{+col:#1}
1311     \cs_if_exist:cTF {\token_to_str:N\color@#1}
1312     {
1313         \convertcolorspec{named}{#1}{HTML}\l_fontspec_hexcol_tl
1314     }
1315     {
1316         \tl_set:Nn \l_fontspec_hexcol_tl {#1}
1317     }
1318 }
1319 \keyval@alias@key[zf]{options}{Color}{Colour}

1320 \newcounter{fontspec_tmp_int}
1321 \define@key[zf]{options}{Opacity}{
1322     \zf@update@family{+opac:#1}
1323     \setcounter {fontspec_tmp_int} { 255*\real{#1} }
1324     \tl_set:Nx \l_fontspec_opacity_tl {
1325         \nhex2 { \value{fontspec_tmp_int} }
1326     }
1327 }

```

Mapping

```

1328 \xetex_or luatex:nnn {
1329     \define@key[zf]{options}{Mapping}
1330 }{
1331     \zf@update@family{+map:#1}
1332     \zf@update@ff{mapping=#1}
1333 }{
1334     \tl_if_eq:nnTF {#1} {tex-text} {
1335         \fontspec_warning:n {no-mapping-ligtex}
1336         \msg_redirect_name:nnn {fontspec} {no-mapping-ligtex} {none}
1337         \setkeys[zf]{options}{ Ligatures=TeX }
1338     }{
1339         \fontspec_warning:n {no-mapping}
1340     }

```

```
1341 }
```

FeatureFile

```
1342 \define@key[zf]{options}{FeatureFile}{
1343   \zf@update@family{+fea:#1}
1344   \zf@update@ff{featurefile=#1}
1345 }
```

23.9.5 Continuous font axes

```
1346 \define@key[zf]{options}{Weight}{
1347   \zf@update@family{+weight:#1}
1348   \zf@update@ff{weight=#1}
1349 }
1350 \define@key[zf]{options}{Width}{
1351   \zf@update@family{+width:#1}
1352   \zf@update@ff{width=#1}
1353 }
1354 \define@key[zf]{options}{OpticalSize}{
1355   \xetex_or luatex:nn {
1356     \ifzf@icu
1357       \tl_set:Nn \l_fontspec_optical_size_tl {/ S = #1}
1358       \zf@update@family{+size:#1}
1359     \fi
1360     \ifzf@mm
1361       \zf@update@family{+size:#1}
1362       \zf@update@ff{optical size=#1}
1363     \fi
1364     \ifzf@icu\else
1365       \ifzf@mm\else
1366         \ifzf@firsttime
1367           \fontspec_warning:n {no-opticals}
1368         \fi
1369       \fi
1370     \fi
1371   }{
1372     \tl_set:Nn \l_fontspec_optical_size_tl {/ S = #1}
1373     \zf@update@family{+size:#1}
1374   }
1375 }
```

23.9.6 Font transformations

These are to be specified to apply directly to a font shape:

```
1376 \define@key[zf]{options}{FakeSlant}[0.2]{
1377   \zf@update@family{+slant:#1}
1378   \zf@update@ff{slant=#1}
1379 }
1380 \define@key[zf]{options}{FakeStretch}[1.2]{
1381   \zf@update@family{+extend:#1}
1382   \zf@update@ff{extend=#1}
1383 }
```

```

1384 \define@key[zf]{options}{FakeBold}[1.5]{
1385   \zf@update@family{+embolden:#1}
1386   \zf@update@ff{embolden=#1}
1387 }

```

These are to be given to a shape that has no real bold/italic to signal that fontspec should automatically create ‘fake’ shapes.

The behaviour is currently that only if both `AutoFakeSlant` *and* `AutoFakeBold` are specified, the bold italic is also faked.

These features presently *override* real shapes found in the font; in the future I’d like these features to be ignored in this case, instead. (This is just a bit harder to program in the current design of fontspec)

```

1388 \define@key[zf]{options}{AutoFakeSlant}[0.2]{
1389   \ifzf@firsttime
1390     \tl_set:Nn \zf@fake@slant {#1}
1391     \tl_put_right:Nn \zf@it@feat {,FakeSlant=#1}
1392     \tl_set_eq:NN \zf@it \zf@fontname
1393     \zf@update@family{fakeit:#1}
1394     \tl_if_empty:NF \zf@fake@embolden {
1395       \tl_put_right:Nx \zf@bfit@feat
1396       {,FakeBold=\zf@fake@embolden,FakeSlant=#1}
1397       \tl_set_eq:NN \zf@bfit \zf@fontname
1398     }
1399   \fi
1400 }

```

Same but reversed:

```

1401 \define@key[zf]{options}{AutoFakeBold}[1.5]{
1402   \ifzf@firsttime
1403     \tl_set:Nn \zf@fake@embolden {#1}
1404     \tl_put_right:Nn \zf@bf@feat {,FakeBold=#1}
1405     \tl_set_eq:NN \zf@bf \zf@fontname
1406     \zf@update@family{fakebf:#1}
1407     \tl_if_empty:NF \zf@fake@slant {
1408       \tl_put_right:Nx \zf@bfit@feat
1409       {,FakeSlant=\zf@fake@slant,FakeBold=#1}
1410       \tl_set_eq:NN \zf@bfit \zf@fontname
1411     }
1412   \fi
1413 }

```

23.9.7 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.

```

1414 \zf@define@font@feature{Ligatures}
1415 \zf@define@feature@option{Ligatures}{Required}      {1}{0}{+rlig}
1416 \zf@define@feature@option{Ligatures}{NoRequired}   {1}{1}{-rlig}
1417 \zf@define@feature@option{Ligatures}{Common}       {1}{2}{+liga}
1418 \zf@define@feature@option{Ligatures}{NoCommon}    {1}{3}{-liga}

```

```

1419 \zf@define@feature@option{Ligatures}{Rare} {1}{4}{+dlig}
1420 \zf@define@feature@option{Ligatures}{NoRare} {1}{5}{-dlig}
1421 \zf@define@feature@option{Ligatures}{Discretionary} {1}{4}{+dlig}
1422 \zf@define@feature@option{Ligatures}{NoDiscretionary}{1}{5}{-dlig}
1423 \zf@define@feature@option{Ligatures}{Contextual} {}{} {+clig}
1424 \zf@define@feature@option{Ligatures}{NoContextual} {}{} {-clig}
1425 \zf@define@feature@option{Ligatures}{Historical} {}{} {+hlig}
1426 \zf@define@feature@option{Ligatures}{NoHistorical} {}{} {-hlig}
1427 \zf@define@feature@option{Ligatures}{Logos} {1}{6} {}
1428 \zf@define@feature@option{Ligatures}{NoLogos} {1}{7} {}
1429 \zf@define@feature@option{Ligatures}{Rebus} {1}{8} {}
1430 \zf@define@feature@option{Ligatures}{NoRebus} {1}{9} {}
1431 \zf@define@feature@option{Ligatures}{Diphthong} {1}{10} {}
1432 \zf@define@feature@option{Ligatures}{NoDiphthong} {1}{11} {}
1433 \zf@define@feature@option{Ligatures}{Squared} {1}{12} {}
1434 \zf@define@feature@option{Ligatures}{NoSquared} {1}{13} {}
1435 \zf@define@feature@option{Ligatures}{AbbrevSquared} {1}{14} {}
1436 \zf@define@feature@option{Ligatures}{NoAbbrevSquared}{1}{15} {}
1437 \zf@define@feature@option{Ligatures}{Icelandic} {1}{32} {}
1438 \zf@define@feature@option{Ligatures}{NoIcelandic} {1}{33} {}

```

Emulate CM extra ligatures.

```

1439 \define@key[zf@feat]{Ligatures}{TeX}[]{}
1440 \xetex_or luatex:nn {
1441   \zf@update@family{+map:tex-text}
1442   \zf@update@ff{mapping=tex-text}
1443 }{
1444   \zf@update@family{+tlig+trep}
1445   \zf@update@ff{+tlig;trep}
1446 }
1447 }

```

23.9.8 Letters

```

1448 \zf@define@font@feature{Letters}
1449 \zf@define@feature@option{Letters}{Normal} {3}{0}{}
1450 \zf@define@feature@option{Letters}{Uppercase} {3}{1}{+case}
1451 \zf@define@feature@option{Letters}{Lowercase} {3}{2}{}
1452 \zf@define@feature@option{Letters}{SmallCaps} {3}{3}{+smcp}
1453 \zf@define@feature@option{Letters}{PetiteCaps} {} {} {+pcap}
1454 \zf@define@feature@option{Letters}{UppercaseSmallCaps} {} {} {+c2sc}
1455 \zf@define@feature@option{Letters}{UppercasePetiteCaps} {} {} {+c2pc}
1456 \zf@define@feature@option{Letters}{InitialCaps} {3}{4}{}
1457 \zf@define@feature@option{Letters}{Unicase} {} {} {+unic}

```

23.9.9 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.

```

1458 \zf@define@font@feature{Numbers}
1459 \zf@define@feature@option{Numbers}{Monospaced} {6} {0}{+tnum}

```

```

1460 \zf@define@feature@option{Numbers}{Proportional} {6} {1}{+pnum}
1461 \zf@define@feature@option{Numbers}{Lowercase} {21}{0}{+onum}
1462 \zf@define@feature@option{Numbers}{OldStyle} {21}{0}{+onum}
1463 \zf@define@feature@option{Numbers}{Uppercase} {21}{1}{+lnum}
1464 \zf@define@feature@option{Numbers}{Lining} {21}{1}{+lnum}
1465 \zf@define@feature@option{Numbers}{SlashedZero} {14}{5}{+zero}
1466 \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.

```

1467 \luatex_if_engine:T {
1468   \zf@define@feature@option{Numbers}{Arabic}{ }{ }{+anum}
1469 }

```

23.9.10 Contextuals

```

1470 \zf@define@font@feature {Contextuals}
1471 \zf@define@feature@option{Contextuals}{Swash} {} {} {+csw}
1472 \zf@define@feature@option{Contextuals}{NoSwash} {} {} {-csw}
1473 \zf@define@feature@option{Contextuals}{Alternate} {} {} {+calt}
1474 \zf@define@feature@option{Contextuals}{NoAlternate} {} {} {-calt}
1475 \zf@define@feature@option{Contextuals}{WordInitial} {8}{0}{+init}
1476 \zf@define@feature@option{Contextuals}{NoWordInitial}{8}{1}{-init}
1477 \zf@define@feature@option{Contextuals}{WordFinal} {8}{2}{+fina}
1478 \zf@define@feature@option{Contextuals}{NoWordFinal} {8}{3}{-fina}
1479 \zf@define@feature@option{Contextuals}{LineInitial} {8}{4}{ }
1480 \zf@define@feature@option{Contextuals}{NoLineInitial}{8}{5}{ }
1481 \zf@define@feature@option{Contextuals}{LineFinal} {8}{6}{+falt}
1482 \zf@define@feature@option{Contextuals}{NoLineFinal} {8}{7}{-falt}
1483 \zf@define@feature@option{Contextuals}{Inner} {8}{8}{+medi}
1484 \zf@define@feature@option{Contextuals}{NoInner} {8}{9}{-medi}

```

23.9.11 Diacritics

```

1485 \zf@define@font@feature{Diacritics}
1486 \zf@define@feature@option{Diacritics}{Show} {9}{0}{ }
1487 \zf@define@feature@option{Diacritics}{Hide} {9}{1}{ }
1488 \zf@define@feature@option{Diacritics}{Decompose} {9}{2}{ }
1489 \zf@define@feature@option{Diacritics}{MarkToBase} {} {} {+mark}
1490 \zf@define@feature@option{Diacritics}{NoMarkToBase} {} {} {-mark}
1491 \zf@define@feature@option{Diacritics}{MarkToMark} {} {} {+mkmk}
1492 \zf@define@feature@option{Diacritics}{NoMarkToMark} {} {} {-mkmk}
1493 \zf@define@feature@option{Diacritics}{AboveBase} {} {} {+abvm}
1494 \zf@define@feature@option{Diacritics}{NoAboveBase} {} {} {-abvm}
1495 \zf@define@feature@option{Diacritics}{BelowBase} {} {} {+blwm}
1496 \zf@define@feature@option{Diacritics}{NoBelowBase} {} {} {-blwm}

```

23.9.12 Kerning

```

1497 \zf@define@font@feature{Kerning}
1498 \zf@define@feature@option{Kerning}{Uppercase} {} {} {+csp}
1499 \zf@define@feature@option{Kerning}{On} {} {} {+kern}

```

```

1500 \zf@define@feature@option{Kerning}{Off}      {}{}{-kern}
1501 %\zf@define@feature@option{Kerning}{Vertical}{}{}{+vkern}
1502 %\zf@define@feature@option{Kerning}
1503 %      {VerticalAlternateProportional}{}{}{+vpal}
1504 %\zf@define@feature@option{Kerning}{VerticalAlternateHalfWidth}{}{}{+vhal}

```

23.9.13 Vertical position

```

1505 \zf@define@font@feature{VerticalPosition}
1506 \zf@define@feature@option{VerticalPosition}{Normal}      {10}{0}{}
1507 \zf@define@feature@option{VerticalPosition}{Superior}    {10}{1}{+sup}
1508 \zf@define@feature@option{VerticalPosition}{Inferior}     {10}{2}{+sub}
1509 \zf@define@feature@option{VerticalPosition}{Ordinal}      {10}{3}{+ordn}
1510 \zf@define@feature@option{VerticalPosition}{Numerator}    {} {} {+numr}
1511 \zf@define@feature@option{VerticalPosition}{Denominator}  {} {} {+dnom}
1512 \zf@define@feature@option{VerticalPosition}{ScientificInferior}{}{}{+sinf}

```

23.9.14 Fractions

```

1513 \zf@define@font@feature{Fractions}
1514 \zf@define@feature@option{Fractions}{On}      {11}{1}{+frac}
1515 \zf@define@feature@option{Fractions}{Off}     {11}{0}{-frac}
1516 \zf@define@feature@option{Fractions}{Diagonal} {11}{2}{}
1517 \zf@define@feature@option{Fractions}{Alternate}{} {} {+afrc}

```

23.9.15 Alternates and variants

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

```

1518 \define@key[zf]{options}{Alternate}[0]{
1519   \clist_set_eq:NN \l_fontspec_tmpa_clist \XKV@rm
1520   \setkeys*[zf@feat]{Alternate}{#1}
1521   \unless\ifx\XKV@rm\@empty
1522     \def\XKV@tfam{Alternate}
1523     \fontspec_make_feature:nnn{17}{#1}{+salt=#1}
1524   \fi
1525   \clist_set_eq:NN \XKV@rm \l_fontspec_tmpa_clist
1526 }

1527 \define@key[zf]{options}{Variant}{
1528   \clist_set_eq:NN \l_fontspec_tmpa_clist \XKV@rm
1529   \setkeys*[zf@feat]{Variant}{#1}
1530   \unless\ifx\XKV@rm\@empty
1531     \def\XKV@tfam{Variant}
1532     \fontspec_make_feature:nnx{18}{#1}{+ss\two@digits{#1}}
1533   \fi
1534   \clist_set_eq:NN \XKV@rm \l_fontspec_tmpa_clist
1535 }
1536 \aliasfontfeature{Variant}{StylisticSet}

```

23.9.16 Style

```

1537 \zf@define@font@feature{Style}
1538 \zf@define@feature@option{Style}{Alternate}      {} {} {+salt}

```

```

1539 \zf@define@feature@option{Style}{Italic} {32}{2}{+ital}
1540 \zf@define@feature@option{Style}{Ruby} {28}{2}{+ruby}
1541 \zf@define@feature@option{Style}{Swash} {} {} {+swsh}
1542 \zf@define@feature@option{Style}{Historic} {} {} {+hist}
1543 \zf@define@feature@option{Style}{Display} {19}{1}{ }
1544 \zf@define@feature@option{Style}{Engraved} {19}{2}{ }
1545 \zf@define@feature@option{Style}{TitlingCaps} {19}{4}{+titl}
1546 \zf@define@feature@option{Style}{TallCaps} {19}{5}{ }
1547 \zf@define@feature@option{Style}{HorizontalKana}{} {} {+hkna}
1548 \zf@define@feature@option{Style}{VerticalKana} {} {} {+vkna}

```

23.9.17 CJK shape

```

1549 \zf@define@font@feature{CJKShape}
1550 \zf@define@feature@option{CJKShape}{Traditional}{20}{0} {+trad}
1551 \zf@define@feature@option{CJKShape}{Simplified} {20}{1} {+smp1}
1552 \zf@define@feature@option{CJKShape}{JIS1978} {20}{2} {+jp78}
1553 \zf@define@feature@option{CJKShape}{JIS1983} {20}{3} {+jp83}
1554 \zf@define@feature@option{CJKShape}{JIS1990} {20}{4} {+jp90}
1555 \zf@define@feature@option{CJKShape}{Expert} {20}{10}{+expt}
1556 \zf@define@feature@option{CJKShape}{NLC} {20}{13}{+nlck}

```

23.9.18 Character width

```

1557 \zf@define@font@feature{CharacterWidth}
1558 \zf@define@feature@option{CharacterWidth}{Proportional}{22}{0}{+pwid}
1559 \zf@define@feature@option{CharacterWidth}{Full}{22}{1}{+fwid}
1560 \zf@define@feature@option{CharacterWidth}{Half}{22}{2}{+hwid}
1561 \zf@define@feature@option{CharacterWidth}{Third}{22}{3}{+twid}
1562 \zf@define@feature@option{CharacterWidth}{Quarter}{22}{4}{+qwid}
1563 \zf@define@feature@option{CharacterWidth}{AlternateProportional}{22}{5}{+palt}
1564 \zf@define@feature@option{CharacterWidth}{AlternateHalf}{22}{6}{+halt}
1565 \zf@define@feature@option{CharacterWidth}{Default}{22}{7}{ }

```

23.9.19 Annotation

```

1566 \zf@define@feature@option{Annotation}{Off}{24}{0}{ }
1567 \zf@define@feature@option{Annotation}{Box}{24}{1}{ }
1568 \zf@define@feature@option{Annotation}{RoundedBox}{24}{2}{ }
1569 \zf@define@feature@option{Annotation}{Circle}{24}{3}{ }
1570 \zf@define@feature@option{Annotation}{BlackCircle}{24}{4}{ }
1571 \zf@define@feature@option{Annotation}{Parenthesis}{24}{5}{ }
1572 \zf@define@feature@option{Annotation}{Period}{24}{6}{ }
1573 \zf@define@feature@option{Annotation}{RomanNumerals}{24}{7}{ }
1574 \zf@define@feature@option{Annotation}{Diamond}{24}{8}{ }
1575 \zf@define@feature@option{Annotation}{BlackSquare}{24}{9}{ }
1576 \zf@define@feature@option{Annotation}{BlackRoundSquare}{24}{10}{ }
1577 \zf@define@feature@option{Annotation}{DoubleCircle}{24}{11}{ }

1578 \define@key[zf]{options}{Annotation}[0]{
1579   \clist_set_eq:NN \l_fontspec_tmpa_clist \XKV@rm
1580   \setkeys*[zf@feat]{Annotation}{#1}
1581   \unless\ifx\XKV@rm@empty
1582     \def\XKV@tfam{Alternate}
1583     \fontspec_make_feature:nnn{ }{+nalt=#1}

```



```

1584 \fi
1585 \clist_set_eq:NN \XKV@rm \l_fontspec_tmpa_clist
1586 }

```

23.9.20 Vertical

```

1587 \zf@define@font@feature{Vertical}
1588 \define@key[zf@feat]{Vertical}{RotatedGlyphs}[]{
1589   \ifzf@icu
1590     \fontspec_make_feature:nnn{}{}{+vrt2}
1591     \zf@update@family{+vert}
1592     \zf@update@eff{vertical}
1593   \else
1594     \zf@update@family{+vert}
1595     \zf@update@eff{vertical}
1596   \fi
1597 }

```

23.9.21 Script

```

1598 \newfontscript{Arabic}{arab}           \newfontscript{Armenian}{armn}
1599 \newfontscript{Balinese}{bali}         \newfontscript{Bengali}{beng}
1600 \newfontscript{Bopomofo}{bopo}        \newfontscript{Braille}{brai}
1601 \newfontscript{Buginese}{bugi}        \newfontscript{Buhid}{buhd}
1602 \newfontscript{Byzantine~Music}{byzm} \newfontscript{Canadian~Syllabics}{cans}
1603 \newfontscript{Cherokee}{cher}
1604 \newfontscript{CJK~Ideographic}{hani} \newfontscript{Coptic}{copt}
1605 \newfontscript{Cypriot~Syllabary}{cpri} \newfontscript{Cyrillic}{cyril}
1606 \newfontscript{Default}{DFLT}         \newfontscript{Deseret}{dsrt}
1607 \newfontscript{Devanagari}{deva}      \newfontscript{Ethiopic}{ethi}
1608 \newfontscript{Georgian}{geor}       \newfontscript{Glagolitic}{glag}
1609 \newfontscript{Gothic}{goth}         \newfontscript{Greek}{grek}
1610 \newfontscript{Gujarati}{gujr}       \newfontscript{Gurmukhi}{guru}
1611 \newfontscript{Hangul~Jamo}{jamo}    \newfontscript{Hangul}{hang}
1612 \newfontscript{Hanunoo}{hano}       \newfontscript{Hebrew}{hebr}
1613 \newfontscript{Hiragana~and~Katakana}{kana}
1614 \newfontscript{Javanese}{java}       \newfontscript{Kannada}{knda}
1615 \newfontscript{Kharosthi}{khar}     \newfontscript{Khmer}{khmr}
1616 \newfontscript{Lao}{lao~}          \newfontscript{Latin}{latn}
1617 \newfontscript{Limbu}{limb}         \newfontscript{Linear~B}{linb}
1618 \newfontscript{Malayalam}{mlym}     \newfontscript{Math}{math}
1619 \newfontscript{Mongolian}{mong}
1620 \newfontscript{Musical~Symbols}{musc} \newfontscript{Myanmar}{mymr}
1621 \newfontscript{N'ko}{nko~}         \newfontscript{Ogham}{ogam}
1622 \newfontscript{Old~Italic}{ital}
1623 \newfontscript{Old~Persian~Cuneiform}{xpeo}
1624 \newfontscript{Oriya}{orya}         \newfontscript{Osmanya}{osma}
1625 \newfontscript{Phags-pa}{phag}      \newfontscript{Phoenician}{phnx}
1626 \newfontscript{Runic}{runr}         \newfontscript{Shavian}{shaw}
1627 \newfontscript{Sinhala}{sinh}
1628 \newfontscript{Sumero-Akkadian~Cuneiform}{xsux}
1629 \newfontscript{Syloti~Nagri}{sylo}  \newfontscript{Syriac}{syrc}
1630 \newfontscript{Tagalog}{tglg}       \newfontscript{Tagbanwa}{tagb}
1631 \newfontscript{Tai~Le}{tale}        \newfontscript{Tai~Lu}{talu}

```

```

1632 \newfontscript{Tamil}{taml}          \newfontscript{Telugu}{telu}
1633 \newfontscript{Thaana}{thaa}         \newfontscript{Thai}{thai}
1634 \newfontscript{Tibetan}{tibT}        \newfontscript{Tifinagh}{tfng}
1635 \newfontscript{Ugaritic~Cuneiform}{ugar}\newfontscript{Yi}{yi~}

```

For convenience:

```

1636 \newfontscript{Kana}{kana}
1637 \newfontscript{Maths}{math}
1638 \newfontscript{CJK}{hani}

```

23.9.22 Language

```

1639 \newfontlanguage{Abaza}{ABA}\newfontlanguage{Abkhazian}{ABK}
1640 \newfontlanguage{Adyghe}{ADY}\newfontlanguage{Afrikaans}{AFK}
1641 \newfontlanguage{Afar}{AFR}\newfontlanguage{Agaw}{AGW}
1642 \newfontlanguage{Altai}{ALT}\newfontlanguage{Amharic}{AMH}
1643 \newfontlanguage{Arabic}{ARA}\newfontlanguage{Aari}{ARI}
1644 \newfontlanguage{Arakanese}{ARK}\newfontlanguage{Assamese}{ASM}
1645 \newfontlanguage{Athapaskan}{ATH}\newfontlanguage{Avar}{AVR}
1646 \newfontlanguage{Awadhi}{AWA}\newfontlanguage{Aymara}{AYM}
1647 \newfontlanguage{Azeri}{AZE}\newfontlanguage{Badaga}{BAD}
1648 \newfontlanguage{Baghelkhandi}{BAG}\newfontlanguage{Balkar}{BAL}
1649 \newfontlanguage{Baule}{BAU}\newfontlanguage{Berber}{BBR}
1650 \newfontlanguage{Bench}{BCH}\newfontlanguage{Bible~Cree}{BCR}
1651 \newfontlanguage{Belarussian}{BEL}\newfontlanguage{Bemba}{BEM}
1652 \newfontlanguage{Bengali}{BEN}\newfontlanguage{Bulgarian}{BGR}
1653 \newfontlanguage{Bhili}{BHI}\newfontlanguage{Bhojpuri}{BHO}
1654 \newfontlanguage{Bikol}{BIK}\newfontlanguage{Bilen}{BIL}
1655 \newfontlanguage{Blackfoot}{BKF}\newfontlanguage{Balochi}{BLI}
1656 \newfontlanguage{Balante}{BLN}\newfontlanguage{Balti}{BLT}
1657 \newfontlanguage{Bambara}{BMB}\newfontlanguage{Bamileke}{BML}
1658 \newfontlanguage{Breton}{BRE}\newfontlanguage{Brahui}{BRH}
1659 \newfontlanguage{Braj~Bhasha}{BRI}\newfontlanguage{Burmese}{BRM}
1660 \newfontlanguage{Bashkir}{BSH}\newfontlanguage{Beti}{BTI}
1661 \newfontlanguage{Catalan}{CAT}\newfontlanguage{Cebuano}{CEB}
1662 \newfontlanguage{Chechen}{CHE}\newfontlanguage{Chaha~Gurage}{CHG}
1663 \newfontlanguage{Chattisgarhi}{CHH}\newfontlanguage{Chichewa}{CHI}
1664 \newfontlanguage{Chukchi}{CHK}\newfontlanguage{Chipewyan}{CHP}
1665 \newfontlanguage{Cherokee}{CHR}\newfontlanguage{Chuvash}{CHU}
1666 \newfontlanguage{Comorian}{CMR}\newfontlanguage{Coptic}{COP}
1667 \newfontlanguage{Cree}{CRE}\newfontlanguage{Carrier}{CRR}
1668 \newfontlanguage{Crimean~Tatar}{CRT}\newfontlanguage{Church~Slavonic}{CSL}
1669 \newfontlanguage{Czech}{CSY}\newfontlanguage{Danish}{DAN}
1670 \newfontlanguage{Dargwa}{DAR}\newfontlanguage{Woods~Cree}{DCR}
1671 \newfontlanguage{German}{DEU}
1672 \newfontlanguage{Dogri}{DGR}\newfontlanguage{Divehi}{DIV}
1673 \newfontlanguage{Djerma}{DJR}\newfontlanguage{Dangme}{DNG}
1674 \newfontlanguage{Dinka}{DNK}\newfontlanguage{Dungan}{DUN}
1675 \newfontlanguage{Dzongkha}{DZN}\newfontlanguage{Ebira}{EBI}
1676 \newfontlanguage{Eastern~Cree}{ECR}\newfontlanguage{Edo}{EDO}
1677 \newfontlanguage{Efik}{EFI}\newfontlanguage{Greek}{ELL}
1678 \newfontlanguage{English}{ENG}\newfontlanguage{Erzya}{ERZ}
1679 \newfontlanguage{Spanish}{ESP}\newfontlanguage{Estonian}{ETI}

```

1680 \newfontlanguage{Basque}{EUQ}\newfontlanguage{Evenki}{EVK}
 1681 \newfontlanguage{Even}{EVN}\newfontlanguage{Ewe}{EWE}
 1682 \newfontlanguage{French~Antillean}{FAN}
 1683 \newfontlanguage{Farsi}{FAR}
 1684 \newfontlanguage{Parsi}{FAR}
 1685 \newfontlanguage{Persian}{FAR}
 1686 \newfontlanguage{Finnish}{FIN}\newfontlanguage{Fijian}{FJI}
 1687 \newfontlanguage{Flemish}{FLE}\newfontlanguage{Forest~Nenets}{FNE}
 1688 \newfontlanguage{Fon}{FON}\newfontlanguage{Faroese}{FOS}
 1689 \newfontlanguage{French}{FRA}\newfontlanguage{Frisian}{FRI}
 1690 \newfontlanguage{Friulian}{FRL}\newfontlanguage{Futa}{FTA}
 1691 \newfontlanguage{Fulani}{FUL}\newfontlanguage{Ga}{GAD}
 1692 \newfontlanguage{Gaelic}{GAE}\newfontlanguage{Gagauz}{GAG}
 1693 \newfontlanguage{Galician}{GAL}\newfontlanguage{Garshuni}{GAR}
 1694 \newfontlanguage{Garhwali}{GAW}\newfontlanguage{Ge'ez}{GEZ}
 1695 \newfontlanguage{Gilyak}{GIL}\newfontlanguage{Gumuz}{GMZ}
 1696 \newfontlanguage{Gondi}{GON}\newfontlanguage{Greenlandic}{GRN}
 1697 \newfontlanguage{Garó}{GRO}\newfontlanguage{Guarani}{GUA}
 1698 \newfontlanguage{Gujarati}{GUJ}\newfontlanguage{Haitian}{HAI}
 1699 \newfontlanguage{Halam}{HAL}\newfontlanguage{Harauti}{HAR}
 1700 \newfontlanguage{Hausa}{HAU}\newfontlanguage{Hawaii}{HAW}
 1701 \newfontlanguage{Hammer-Banna}{HBN}\newfontlanguage{Hiligaynon}{HIL}
 1702 \newfontlanguage{Hindi}{HIN}\newfontlanguage{High~Mari}{HMA}
 1703 \newfontlanguage{Hindko}{HND}\newfontlanguage{Ho}{HO}
 1704 \newfontlanguage{Harari}{HRI}\newfontlanguage{Croatian}{HRV}
 1705 \newfontlanguage{Hungarian}{HUN}\newfontlanguage{Armenian}{HYE}
 1706 \newfontlanguage{Igbo}{IBO}\newfontlanguage{Ijo}{IJO}
 1707 \newfontlanguage{Ilokano}{ILO}\newfontlanguage{Indonesian}{IND}
 1708 \newfontlanguage{Ingush}{ING}\newfontlanguage{Inuktitut}{INU}
 1709 \newfontlanguage{Irish}{IRI}\newfontlanguage{Irish~Traditional}{IRT}
 1710 \newfontlanguage{Icelandic}{ISL}\newfontlanguage{Inari~Sami}{ISM}
 1711 \newfontlanguage{Italian}{ITA}\newfontlanguage{Hebrew}{IWR}
 1712 \newfontlanguage{Javanese}{JAV}\newfontlanguage{Yiddish}{JII}
 1713 \newfontlanguage{Japanese}{JAN}\newfontlanguage{Judezmo}{JUD}
 1714 \newfontlanguage{Jula}{JUL}\newfontlanguage{Kabardian}{KAB}
 1715 \newfontlanguage{Kachchi}{KAC}\newfontlanguage{Kalenjin}{KAL}
 1716 \newfontlanguage{Kannada}{KAN}\newfontlanguage{Karachay}{KAR}
 1717 \newfontlanguage{Georgian}{KAT}\newfontlanguage{Kazakh}{KAZ}
 1718 \newfontlanguage{Kebena}{KEB}\newfontlanguage{Khutsuri~Georgian}{KGE}
 1719 \newfontlanguage{Khakass}{KHA}\newfontlanguage{Khanty-Kazim}{KHK}
 1720 \newfontlanguage{Khmer}{KHM}\newfontlanguage{Khanty-Shurishkar}{KHS}
 1721 \newfontlanguage{Khanty-Vakhi}{KHV}\newfontlanguage{Khowar}{KHW}
 1722 \newfontlanguage{Kikuyu}{KIK}\newfontlanguage{Kirghiz}{KIR}
 1723 \newfontlanguage{Kisii}{KIS}\newfontlanguage{Kokni}{KKN}
 1724 \newfontlanguage{Kalmyk}{KLM}\newfontlanguage{Kamba}{KMB}
 1725 \newfontlanguage{Kumaoni}{KMN}\newfontlanguage{Komo}{KMO}
 1726 \newfontlanguage{Komso}{KMS}\newfontlanguage{Kanuri}{KNR}
 1727 \newfontlanguage{Kodagu}{KOD}\newfontlanguage{Korean~Old~Hangul}{KOH}
 1728 \newfontlanguage{Konkani}{KOK}\newfontlanguage{Kikongo}{KON}
 1729 \newfontlanguage{Komi-Permyak}{KOP}\newfontlanguage{Korean}{KOR}
 1730 \newfontlanguage{Komi-Zyrian}{KOZ}\newfontlanguage{Kpelle}{KPL}

1731 \newfontlanguage{Krio}{KRI}\newfontlanguage{Karakalpak}{KPK}
1732 \newfontlanguage{Karelian}{KRL}\newfontlanguage{Karaim}{KRM}
1733 \newfontlanguage{Karen}{KRN}\newfontlanguage{Koorete}{KRT}
1734 \newfontlanguage{Kashmiri}{KSH}\newfontlanguage{Khasi}{KSI}
1735 \newfontlanguage{Kildin~Sami}{KSM}\newfontlanguage{Kui}{KUI}
1736 \newfontlanguage{Kulvi}{KUL}\newfontlanguage{Kumyk}{KUM}
1737 \newfontlanguage{Kurdish}{KUR}\newfontlanguage{Kurukh}{KUU}
1738 \newfontlanguage{Kuy}{KUY}\newfontlanguage{Koryak}{KYK}
1739 \newfontlanguage{Ladin}{LAD}\newfontlanguage{Lahuli}{LAH}
1740 \newfontlanguage{Lak}{LAK}\newfontlanguage{Lambani}{LAM}
1741 \newfontlanguage{Lao}{LAO}\newfontlanguage{Latin}{LAT}
1742 \newfontlanguage{Laz}{LAZ}\newfontlanguage{L-Cree}{LCR}
1743 \newfontlanguage{Ladakhi}{LDK}\newfontlanguage{Lezgi}{LEZ}
1744 \newfontlanguage{Lingala}{LIN}\newfontlanguage{Low~Mari}{LMA}
1745 \newfontlanguage{Limbu}{LMB}\newfontlanguage{Lomwe}{LMW}
1746 \newfontlanguage{Lower~Sorbian}{LSB}\newfontlanguage{Lule~Sami}{LSM}
1747 \newfontlanguage{Lithuanian}{LTH}\newfontlanguage{Luba}{LUB}
1748 \newfontlanguage{Luganda}{LUG}\newfontlanguage{Luhya}{LUH}
1749 \newfontlanguage{Luo}{LUO}\newfontlanguage{Latvian}{LVI}
1750 \newfontlanguage{Majang}{MAJ}\newfontlanguage{Makua}{MAK}
1751 \newfontlanguage{Malayalam~Traditional}{MAL}\newfontlanguage{Mansi}{MAN}
1752 \newfontlanguage{Marathi}{MAR}\newfontlanguage{Marwari}{MAW}
1753 \newfontlanguage{Mbundu}{MBN}\newfontlanguage{Manchu}{MCH}
1754 \newfontlanguage{Moose~Cree}{MCR}\newfontlanguage{Mende}{MDE}
1755 \newfontlanguage{Me'en}{MEN}\newfontlanguage{Mizo}{MIZ}
1756 \newfontlanguage{Macedonian}{MKD}\newfontlanguage{Male}{MLE}
1757 \newfontlanguage{Malagasy}{MLG}\newfontlanguage{Malinke}{MLN}
1758 \newfontlanguage{Malayalam~Reformed}{MLR}\newfontlanguage{Malay}{MLY}
1759 \newfontlanguage{Mandinka}{MND}\newfontlanguage{Mongolian}{MNG}
1760 \newfontlanguage{Manipuri}{MNI}\newfontlanguage{Maninka}{MNK}
1761 \newfontlanguage{Manx~Gaelic}{MNX}\newfontlanguage{Moksha}{MOK}
1762 \newfontlanguage{Moldavian}{MOL}\newfontlanguage{Mon}{MON}
1763 \newfontlanguage{Moroccan}{MOR}\newfontlanguage{Maori}{MRI}
1764 \newfontlanguage{Maithili}{MTH}\newfontlanguage{Maltese}{MTS}
1765 \newfontlanguage{Mundari}{MUN}\newfontlanguage{Naga-Assamese}{NAG}
1766 \newfontlanguage{Nanai}{NAN}\newfontlanguage{Naskapi}{NAS}
1767 \newfontlanguage{N-Cree}{NCR}\newfontlanguage{Ndebele}{NDB}
1768 \newfontlanguage{Ndonga}{NDG}\newfontlanguage{Nepali}{NEP}
1769 \newfontlanguage{Newari}{NEW}\newfontlanguage{Nagari}{NGR}
1770 \newfontlanguage{Norway~House~Cree}{NHC}\newfontlanguage{Nisi}{NIS}
1771 \newfontlanguage{Niuean}{NIU}\newfontlanguage{Nkole}{NKL}
1772 \newfontlanguage{N'ko}{NKO}\newfontlanguage{Dutch}{NLD}
1773 \newfontlanguage{Nogai}{NOG}\newfontlanguage{Norwegian}{NOR}
1774 \newfontlanguage{Northern~Sami}{NSM}\newfontlanguage{Northern~Tai}{NTA}
1775 \newfontlanguage{Esperanto}{NTO}\newfontlanguage{Nynorsk}{NYN}
1776 \newfontlanguage{Oji-Cree}{OCR}\newfontlanguage{Ojibway}{OBJ}
1777 \newfontlanguage{Oriya}{ORI}\newfontlanguage{Oromo}{ORO}
1778 \newfontlanguage{Ossetian}{OSS}\newfontlanguage{Palestinian~Aramaic}{PAA}
1779 \newfontlanguage{Pali}{PAL}\newfontlanguage{Punjabi}{PAN}
1780 \newfontlanguage{Palpa}{PAP}\newfontlanguage{Pashto}{PAS}
1781 \newfontlanguage{Polytonic~Greek}{PGR}\newfontlanguage{Pilipino}{PIL}

1782 \newfontlanguage{Palaung}{PLG}\newfontlanguage{Polish}{PLK}
1783 \newfontlanguage{Provençal}{PRO}\newfontlanguage{Portuguese}{PTG}
1784 \newfontlanguage{Chin}{QIN}\newfontlanguage{Rajasthani}{RAJ}
1785 \newfontlanguage{R-Cree}{RCR}\newfontlanguage{Russian~Buriat}{RBU}
1786 \newfontlanguage{Riang}{RIA}\newfontlanguage{Rhaeto-Romanic}{RMS}
1787 \newfontlanguage{Romanian}{ROM}\newfontlanguage{Romany}{ROY}
1788 \newfontlanguage{Rusyn}{RSY}\newfontlanguage{Ruanda}{RUA}
1789 \newfontlanguage{Russian}{RUS}\newfontlanguage{Sadri}{SAD}
1790 \newfontlanguage{Sanskrit}{SAN}\newfontlanguage{Santali}{SAT}
1791 \newfontlanguage{Sayisi}{SAY}\newfontlanguage{Sekota}{SEK}
1792 \newfontlanguage{Selkup}{SEL}\newfontlanguage{Sango}{SGO}
1793 \newfontlanguage{Shan}{SHN}\newfontlanguage{Sibe}{SIB}
1794 \newfontlanguage{Sidamo}{SID}\newfontlanguage{Silte~Gurage}{SIG}
1795 \newfontlanguage{Skolt~Sami}{SKS}\newfontlanguage{Slovak}{SKY}
1796 \newfontlanguage{Slavey}{SLA}\newfontlanguage{Slovenian}{SLV}
1797 \newfontlanguage{Somali}{SML}\newfontlanguage{Samoan}{SMO}
1798 \newfontlanguage{Sena}{SNA}\newfontlanguage{Sindhi}{SND}
1799 \newfontlanguage{Sinhalese}{SNH}\newfontlanguage{Soninke}{SNK}
1800 \newfontlanguage{Sodo~Gurage}{SOG}\newfontlanguage{Sotho}{SOT}
1801 \newfontlanguage{Albanian}{SQI}\newfontlanguage{Serbian}{SRB}
1802 \newfontlanguage{Saraiki}{SRK}\newfontlanguage{Serer}{SRR}
1803 \newfontlanguage{South~Slavey}{SSL}\newfontlanguage{Southern~Sami}{SSM}
1804 \newfontlanguage{Suri}{SUR}\newfontlanguage{Svan}{SVA}
1805 \newfontlanguage{Swedish}{SVE}\newfontlanguage{Swadaya~Aramaic}{SWA}
1806 \newfontlanguage{Swahili}{SWK}\newfontlanguage{Swazi}{SWZ}
1807 \newfontlanguage{Sutu}{SXT}\newfontlanguage{Syriac}{SYR}
1808 \newfontlanguage{Tabasaran}{TAB}\newfontlanguage{Tajiki}{TAJ}
1809 \newfontlanguage{Tamil}{TAM}\newfontlanguage{Tatar}{TAT}
1810 \newfontlanguage{TH-Cree}{TCR}\newfontlanguage{Telugu}{TEL}
1811 \newfontlanguage{Tongan}{TGN}\newfontlanguage{Tigre}{TGR}
1812 \newfontlanguage{Tigrinya}{TGY}\newfontlanguage{Thai}{THA}
1813 \newfontlanguage{Tahitian}{THT}\newfontlanguage{Tibetan}{TIB}
1814 \newfontlanguage{Turkmen}{TKM}\newfontlanguage{Temne}{TMN}
1815 \newfontlanguage{Tswana}{TNA}\newfontlanguage{Tundra~Nenets}{TNE}
1816 \newfontlanguage{Tonga}{TNG}\newfontlanguage{Todo}{TOD}
1817 \newfontlanguage{Tsonga}{TSG}\newfontlanguage{Turoyo~Aramaic}{TUA}
1818 \newfontlanguage{Tulu}{TUL}\newfontlanguage{Tuvini}{TUV}
1819 \newfontlanguage{Twi}{TWI}\newfontlanguage{Udmurt}{UDM}
1820 \newfontlanguage{Ukrainian}{UKR}\newfontlanguage{Urdu}{URD}
1821 \newfontlanguage{Upper~Sorbian}{USB}\newfontlanguage{Uyghur}{UYG}
1822 \newfontlanguage{Uzbek}{UZB}\newfontlanguage{Venda}{VEN}
1823 \newfontlanguage{Vietnamese}{VIT}\newfontlanguage{Wa}{WA}
1824 \newfontlanguage{Wagdi}{WAG}\newfontlanguage{West-Cree}{WCR}
1825 \newfontlanguage{Welsh}{WEL}\newfontlanguage{Wolof}{WLF}
1826 \newfontlanguage{Tai~Lue}{XBD}\newfontlanguage{Xhosa}{XHS}
1827 \newfontlanguage{Yakut}{YAK}\newfontlanguage{Yoruba}{YBA}
1828 \newfontlanguage{Y-Cree}{YCR}\newfontlanguage{Yi~Classic}{YIC}
1829 \newfontlanguage{Yi~Modern}{YIM}\newfontlanguage{Chinese~Hong~Kong}{ZHH}
1830 \newfontlanguage{Chinese~Phonetic}{ZHP}\newfontlanguage{Chinese~Simplified}{ZHS}
1831 \newfontlanguage{Chinese~Traditional}{ZHT}\newfontlanguage{Zande}{ZND}
1832 \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:

```

1833 \define@key[zf@feat]{Lang}{Turkish}[]{}
1834 \fontspec_check_lang:nTF {TRK} {
1835   \c@zf@language\l_fontspec_strnum_int\relax
1836   \zf@update@family{+lang=Turkish}
1837   \tl_set:Nn \l_fontspec_lang_tl {TRK}
1838 }{
1839   \fontspec_check_lang:nTF {TUR} {
1840     \c@zf@language\l_fontspec_strnum_int\relax
1841     \zf@update@family{+lang=Turkish}
1842     \tl_set:Nn \l_fontspec_lang_tl {TUR}
1843   }{
1844     \fontspec_warning:nx {language-not-exist} {#1}
1845   }
1846 }
1847 }

```

Default

```

1848 \define@key[zf@feat]{Lang}{Default}[]{}
1849 \zf@update@family{+lang=dflt}
1850 \tl_set:Nn \l_fontspec_lang_tl {DFLT}
1851 \c@zf@language=0\relax
1852 }

```

23.9.23 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.

```

1853 \define@key[zf]{options}{RawFeature}{
1854   \zf@update@family{+Raw:#1}
1855   \zf@update@ff{#1}
1856 }

```

23.10 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.

```

1857 \providecommand*{\sidefault}{si}
1858 \DeclareRobustCommand{\sishape}{
1859   \not@math@alphabet\sishape\relax
1860   \fontshape\sidefault\selectfont
1861 }

```

```

1862 \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.

1863 \newcommand*{\zf@merge@shape}[3]{
1864   \edef\@tempa{#1}
1865   \edef\@tempb{#2}
1866   \ifx\@f@shape\@tempb
1867     \ifcsname\@f@encoding/\@f@family/\@f@series/#3\endcsname
1868       \edef\@tempa{#3}
1869     \fi
1870   \fi
1871   \fontshape{\@tempa}\selectfont
1872 }

\itshape Here the original \. . shape commands are redefined to use the merge shape
\scshape macro.
\upshape 1873 \DeclareRobustCommand \itshape {
1874   \not@math@alphabet\itshape\mathit
1875   \zf@merge@shape\itdefault\scdefault\sidefault
1876 }
1877 \DeclareRobustCommand \slshape {
1878   \not@math@alphabet\slshape\relax
1879   \zf@merge@shape\sldefault\scdefault\sidefault
1880 }
1881 \DeclareRobustCommand \scshape {
1882   \not@math@alphabet\scshape\relax
1883   \zf@merge@shape\scdefault\itdefault\sidefault
1884 }
1885 \DeclareRobustCommand \upshape {
1886   \not@math@alphabet\upshape\relax
1887   \zf@merge@shape\updefault\sidefault\scdefault
1888 }

```

23.11 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 at-
          tempt. 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.

1889 \ifpackageloaded{euler}{\zf@package@euler@loadedtrue}
1890                          {\zf@package@euler@loadedfalse}

```

```

1891 \def\zf@math{
1892   \let\zf@font@warning\@font@warning
1893   \let\@font@warning\@font@info
1894   \@ifpackageloaded{euler}{
1895     \ifzf@package@euler@loaded
1896       \zf@math@eulertrue
1897   \else
1898     \fontspec_error:n {euler-too-late}
1899   \fi
1900 }{}
1901 \@ifpackageloaded{lucbmath}{\zf@math@lucidatrue{}}
1902 \@ifpackageloaded{lucidabr}{\zf@math@lucidatrue{}}
1903 \@ifpackageloaded{lucimatx}{\zf@math@lucidatrue{}}

```

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

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

```

1904 \DeclareSymbolFont{legacymaths}{OT1}{cmr}{m}{n}
1905 \SetSymbolFont{legacymaths}{bold}{OT1}{cmr}{bx}{n}
1906 \DeclareMathAccent{\acute}{\mathalpha}{legacymaths}{19}
1907 \DeclareMathAccent{\grave}{\mathalpha}{legacymaths}{18}
1908 \DeclareMathAccent{\ddot}{\mathalpha}{legacymaths}{127}
1909 \DeclareMathAccent{\tilde}{\mathalpha}{legacymaths}{126}
1910 \DeclareMathAccent{\bar}{\mathalpha}{legacymaths}{22}
1911 \DeclareMathAccent{\breve}{\mathalpha}{legacymaths}{21}
1912 \DeclareMathAccent{\check}{\mathalpha}{legacymaths}{20}
1913 \DeclareMathAccent{\hat}{\mathalpha}{legacymaths}{94} % too bad, euler
1914 \DeclareMathAccent{\dot}{\mathalpha}{legacymaths}{95}
1915 \DeclareMathAccent{\mathring}{\mathalpha}{legacymaths}{23}

```

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

```

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

% amsmath.sty:
\renewcommand{\colon}{\nobreak\mskip2mu\mathpunct{}\nonscript
\mkern-\thinmuskip{:}\mskip6mu\plus1mu\relax}

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

```



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

(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.

```
1916 \begingroup
1917   \mathchardef\@tempa="603A \relax
1918   \let\next\egroup
1919   \ifx\colon\@tempa
1920     \DeclareMathSymbol{\colon}{\mathpunct}{legacymaths}{58}
1921   \fi
1922 \endgroup
```

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

```
1923 \ifzf@math@euler\else
1924   \DeclareMathSymbol{!}{\mathclose}{legacymaths}{33}
1925   \DeclareMathSymbol{:}{\mathrel}{legacymaths}{58}
1926   \DeclareMathSymbol{;}{\mathpunct}{legacymaths}{59}
1927   \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.

```
1928 \ifzf@math@lucida\else
1929   \DeclareMathSymbol{0}{\mathalpha}{legacymaths}{‘0}
1930   \DeclareMathSymbol{1}{\mathalpha}{legacymaths}{‘1}
1931   \DeclareMathSymbol{2}{\mathalpha}{legacymaths}{‘2}
1932   \DeclareMathSymbol{3}{\mathalpha}{legacymaths}{‘3}
1933   \DeclareMathSymbol{4}{\mathalpha}{legacymaths}{‘4}
1934   \DeclareMathSymbol{5}{\mathalpha}{legacymaths}{‘5}
1935   \DeclareMathSymbol{6}{\mathalpha}{legacymaths}{‘6}
1936   \DeclareMathSymbol{7}{\mathalpha}{legacymaths}{‘7}
1937   \DeclareMathSymbol{8}{\mathalpha}{legacymaths}{‘8}
1938   \DeclareMathSymbol{9}{\mathalpha}{legacymaths}{‘9}
1939   \DeclareMathSymbol{\Gamma}{\mathalpha}{legacymaths}{0}
1940   \DeclareMathSymbol{\Delta}{\mathalpha}{legacymaths}{1}
1941   \DeclareMathSymbol{\Theta}{\mathalpha}{legacymaths}{2}
1942   \DeclareMathSymbol{\Lambda}{\mathalpha}{legacymaths}{3}
1943   \DeclareMathSymbol{\Xi}{\mathalpha}{legacymaths}{4}
1944   \DeclareMathSymbol{\Pi}{\mathalpha}{legacymaths}{5}
1945   \DeclareMathSymbol{\Sigma}{\mathalpha}{legacymaths}{6}
1946   \DeclareMathSymbol{\Upsilon}{\mathalpha}{legacymaths}{7}
1947   \DeclareMathSymbol{\Phi}{\mathalpha}{legacymaths}{8}
1948   \DeclareMathSymbol{\Psi}{\mathalpha}{legacymaths}{9}
1949   \DeclareMathSymbol{\Omega}{\mathalpha}{legacymaths}{10}
1950   \DeclareMathSymbol{+}{\mathbin}{legacymaths}{43}
```

```

1951 \DeclareMathSymbol{=}{\mathrel}{legacymaths}{61}
1952 \DeclareMathDelimiter{()}{\mathopen}{legacymaths}{40}{largesymbols}{0}
1953 \DeclareMathDelimiter{)}{\mathclose}{legacymaths}{41}{largesymbols}{1}
1954 \DeclareMathDelimiter{[]}{\mathopen}{legacymaths}{91}{largesymbols}{2}
1955 \DeclareMathDelimiter{[]}{\mathclose}{legacymaths}{93}{largesymbols}{3}
1956 \DeclareMathDelimiter{/}{\mathord}{legacymaths}{47}{largesymbols}{14}
1957 \DeclareMathSymbol{\mathdollar}{\mathord}{legacymaths}{36}
1958 \fi
1959 \fi

```

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

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

```

1960 \DeclareSymbolFont{operators}\zf@enc\zf@rmmaths\mddefault\updefault
1961 \SetSymbolFont{operators}{normal}\zf@enc\zf@rmmaths\mddefault\updefault
1962 \SetMathAlphabet\mathrm{normal}\zf@enc\zf@rmmaths\mddefault\updefault
1963 \SetMathAlphabet\mathit{normal}\zf@enc\zf@rmmaths\mddefault\itdefault
1964 \SetMathAlphabet\mathbf{normal}\zf@enc\zf@rmmaths\bfdefault\updefault
1965 \SetMathAlphabet\mathsf{normal}\zf@enc\zf@sffmaths\mddefault\updefault
1966 \SetMathAlphabet\mathtt{normal}\zf@enc\zf@ttmaths\mddefault\updefault
1967 \SetSymbolFont{operators}{bold}\zf@enc\zf@rmmaths\bfdefault\updefault
1968 \ifdefined\zf@rmboldmaths
1969 \SetMathAlphabet\mathrm{bold}\zf@enc\zf@rmboldmaths\mddefault\updefault
1970 \SetMathAlphabet\mathbf{bold}\zf@enc\zf@rmboldmaths\bfdefault\updefault
1971 \SetMathAlphabet\mathit{bold}\zf@enc\zf@rmboldmaths\mddefault\itdefault
1972 \else
1973 \SetMathAlphabet\mathrm{bold}\zf@enc\zf@rmmaths\bfdefault\updefault
1974 \SetMathAlphabet\mathit{bold}\zf@enc\zf@rmmaths\bfdefault\itdefault
1975 \fi
1976 \SetMathAlphabet\mathsf{bold}\zf@enc\zf@sffmaths\bfdefault\updefault
1977 \SetMathAlphabet\mathtt{bold}\zf@enc\zf@ttmaths\bfdefault\updefault
1978 \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 'L^AT_EX Font Catalogue': <http://www.tug.dk/FontCatalogue/mathfonts.html>. I'm sure there are more I've missed. Do the T_EX 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.

```

1979 \def\zf@math@maybe{
1980 \ifpackageloaded{anttor}{
1981 \ifx\define@antt@mathversions a\@zf@mathfalse\fi}{
1982 \ifpackageloaded{arev}{\@zf@mathfalse}{
1983 \ifpackageloaded{eulervm}{\@zf@mathfalse}{
1984 \ifpackageloaded{mathdesign}{\@zf@mathfalse}{
1985 \ifpackageloaded{concmath}{\@zf@mathfalse}{
1986 \ifpackageloaded{cmbright}{\@zf@mathfalse}{
1987 \ifpackageloaded{mathesf}{\@zf@mathfalse}{

```

```

1988 \ifpackageloaded{gfsartemisia}{\@zf@mathfalse}{}
1989 \ifpackageloaded{gfsneohellenic}{\@zf@mathfalse}{}
1990 \ifpackageloaded{iwona}{
1991   \ifx\define@iwona@mathversions a\@zf@mathfalse\fi}{}
1992 \ifpackageloaded{kpfonts}{\@zf@mathfalse}{}
1993 \ifpackageloaded{kmath}{\@zf@mathfalse}{}
1994 \ifpackageloaded{kurier}{
1995   \ifx\define@kurier@mathversions a\@zf@mathfalse\fi}{}
1996 \ifpackageloaded{fouriernc}{\@zf@mathfalse}{}
1997 \ifpackageloaded{fourier}{\@zf@mathfalse}{}
1998 \ifpackageloaded{mathpazo}{\@zf@mathfalse}{}
1999 \ifpackageloaded{mathptmx}{\@zf@mathfalse}{}
2000 \ifpackageloaded{MinionPro}{\@zf@mathfalse}{}
2001 \ifpackageloaded{unicode-math}{\@zf@mathfalse}{}
2002 \ifpackageloaded{breqn}{\@zf@mathfalse}{}
2003 \if@zf@math
2004   \fontspec_info:n {setup-math}
2005   \zf@math
2006 \fi
2007 }
2008 \AtBeginDocument{\@zf@math@maybe}

```

23.12 Finishing up

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

```

2009 \if@zf@configfile
2010   \InputIfFileExists{fontspec.cfg}
2011   {\typeout{fontspec.cfg~ loaded.}}
2012   {\typeout{No~ fontspec.cfg~ file~ found;~ no~ configuration~ loaded.}}
2013 \fi

```

The end! Thanks for coming.

Part VIII

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][1] 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][1] == 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

```

The following are the function that get called from \TeX end.

```

85
86 local function tempwatrue() fontspec.sprint([[ \@tempwatrue]]) end
87 local function tempwafalse() fontspec.sprint([[ \@tempwafalse]]) end
88
89 function fontspec.check_ot_script(fnt, script)
90     if check_script(font.id(fnt), script) then
91         tempwatrue()

```

```

92     else
93         tempswafalse()
94     end
95 end
96
97 function fontspec.check_ot_lang(fnt, lang, script)
98     if check_language(font.id(fnt), lang, script) then
99         tempswattrue()
100     else
101         tempswafalse()
102     end
103 end
104
105 function fontspec.check_ot_feat(fnt, feat, lang, script)
106     for _, f in ipairs { "+trep", "+tlig", "+anum" } do
107         if feat == f then
108             tempswattrue()
109             return
110         end
111     end
112     if check_feature(font.id(fnt), feat, lang, script) then
113         tempswattrue()
114     else
115         tempswafalse()
116     end
117 end
118

```

Part IX

fontspec-patches.sty

```
1 \ExplSyntaxOn
```

23.13 Unicode footnote symbols

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

23.14 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 {
\eminnershape 4   \@nomath\em
5   \tl_if_eq:xxTF \f@shape \itdefault \eminnershape \emshape
6 }
7 \DeclareTextFontCommand{\emph}{\em}
8 \let\emshape\itshape
9 \let\eminnershape\upshape
```

23.15 \-

\- 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{'\-}
```

23.16 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 X

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
```

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	
\-	10, 863
\@sverb	40, 42
\@empty	289, 293, 374, 389, 587, 639, 650, 660, 671–673, 691, 692, 728, 740, 743, 821, 823, 870, 875, 921, 968, 978, 983, 986, 991, 997, 998, 1137, 1148, 1171, 1177, 1211, 1215, 1262, 1521, 1530, 1581
\@firstofone	771
\@font@info	1893
\@font@warning	1892, 1893
\@gobble	784
\@ifpackageloaded	50, 59, 69, 74, 1889, 1894, 1901–1903, 1980, 1982–1990, 1992–1994, 1996–2002
\@ifstar	40
\@makeoother	38
\@namedef	51, 55, 60
\@nameuse	591
\@ne	580, 1023, 1043, 1067
\@noligs	39
\@nomath	4
\@onlypreamble	270–273
\@sverb	42
\@sxverbatim	55
\@tempa	46, 389, 585, 587, 727, 728, 730, 734, 875, 876, 921, 925, 967, 968, 984, 986, 1136, 1137, 1147, 1148, 1170, 1171, 1261, 1262, 1864, 1868, 1871, 1917, 1919
\@tempb	733, 734, 970, 973, 975, 978, 979, 983, 984, 1865, 1866
\@tempcntb	1016, 1018, 1021, 1035, 1038, 1041, 1057, 1061, 1065
\@tempdima	1234, 1236, 1263–1265, 1267, 1273, 1280, 1281
\@tempdimb	1235, 1236, 1264, 1268, 1274
\@tempdimc	1236, 1237, 1265, 1269, 1275
\@tempfonta	729, 730
\@tempfontb	732, 733, 738
\@tempswafalse	87, 667–669, 1017, 1037, 1060
\@tempswatru	86, 666, 1020, 1040, 1064
\@verb	40
\@verbatim	52, 55, 65
\@zf@configfilefalse	214
\@zf@configfiletrue	213
\@zf@mathfalse	212, 1981–1989, 1991–1993, 1995–2002
\@zf@mathtrue	211
\	75, 91, 102–104, 112, 120, 125, 138, 143, 148, 153, 154, 163, 164, 181
\~	32, 33
A	
\active	32, 33
\acute	1906
\addfontfeature	125, 306
\addfontfeatures	290
\advance	62, 579, 1023, 1043, 1067
\aliasfontfeature	334, 1091, 1536
\aliasfontfeatureoption	334
\AtBeginDocument	43, 2008
B	
\bar	1910
\begingroup	31, 52, 292, 526, 726, 1916
\bfdefault	641, 644, 675, 678, 682, 686, 694, 698, 1964, 1967, 1970, 1973, 1974, 1976, 1977
\bgroup	37
\bool_if:N	1094
\bool_if:NT	669
\bool_if:N	569
\bool_if:n	1002
\bool_new:N	1081
\bool_set_false:N	565
\bool_set_true:N	567, 1085
\breve	1911
C	
\c@zf@index	42, 1017–1019, 1021, 1023, 1036, 1038, 1039, 1041, 1043, 1059, 1061, 1063, 1065, 1067
\c@zf@language	44, 366, 421, 439, 629, 1057, 1062, 1835, 1840, 1851
\c@zf@newff	41, 309
\c@zf@script	43, 349, 419, 438, 467, 483, 627, 1035, 1039, 1057, 1062
\c_minus_one	1291

450, 462, 463, 478, 479, 493, 494, 507, 508, 577, 578, 580, 583, 588, 1867	\fontspec_calc_scale:n 1224, 1225, 1231
\endgroup 34, 299, 546, 754, 1922	\fontspec_check_lang:n 1032
environments:	\fontspec_check_lang:nTF 363, 469, 484, 1032 , 1834, 1839
listingcont* 58	\fontspec_check_ot_feat:n 1055
verbatim* 49	\fontspec_check_ot_feat:nTF 880, 884, 1054
\etex_iffontchar:D 50	\fontspec_check_ot_feat:nTF 425, 440, 935, 1054
\ExecuteOptions 223	\fontspec_check_script:n 1013
\exp_after:wN 1008, 1009	\fontspec_check_script:nTF 346, 453, 598, 1013
\exp_not:N . 58, 62, 280, 281, 295, 786, 792	\fontspec_complete_fontname:Nn 1132, 1142, 1153, 1158, 1162, 1166, 1176, 1180 , 1219
\exp_not:V 826	\fontspec_declare_shape:nnnn 739, 744, 750, 756
\expandafter 20, 22, 579, 582, 587, 588, 771	\fontspec_error:n 64, 772, 1898
\ExplSyntaxOn 1, 3	\fontspec_error:nx 65, 826, 1306
F	\fontspec_fullname:n 556, 558, 571 , 624, 729, 732, 762, 776
\f@encoding 1867	\fontspec_get_features:n 533, 759, 773, 820
\f@family 241, 291, 296, 297, 378, 385, 386, 402, 403, 415, 416, 434, 435, 449, 450, 462, 463, 478, 479, 493, 494, 507, 508, 1867	\fontspec_glyph_if_exist:Nn 49
\f@series 27, 1867	\fontspec_glyph_if_exist:NnTF 49 , 1297
\f@shape 5, 27, 1866	\fontspec_if_aat_feature:nn 384
\f@size 386, 403, 416, 435, 450, 463, 479, 494, 508, 556, 558, 729, 732	\fontspec_if_aat_feature:nnTF .. 1 , 384
\FancyVerbSpace 70	\fontspec_if_current_language:n .. 506
\fi . 16, 23, 36, 303, 318, 327, 382, 393, 396, 399, 409, 412, 428, 431, 443, 446, 456, 459, 472, 475, 487, 490, 501, 504, 515, 518, 584, 590, 618, 645, 646, 656, 657, 663, 667, 668, 679, 683, 687, 688, 695, 699, 710, 713, 719, 731, 745–747, 751–753, 805, 818, 877, 878, 881, 897, 904, 907, 926, 980–982, 987, 988, 997, 998, 1024, 1044, 1068, 1144, 1155, 1178, 1210, 1258, 1270, 1359, 1363, 1368–1370, 1399, 1412, 1524, 1533, 1584, 1596, 1869, 1870, 1899, 1921, 1958, 1959, 1975, 1981, 1991, 1995, 2006, 2013	\fontspec_if_current_language:nTF 1 , 506
\fi: 54, 1026, 1030, 1046, 1052, 1070, 1079	\fontspec_if_current_script:n 492
\font 12, 15, 19, 386, 403, 416, 435, 450, 463, 479, 494, 508, 556, 558, 729, 732, 1234, 1246, 1273–1275, 1281, 1291, 1300, 1302	\fontspec_if_current_script:nTF 1 , 492
\font@warning 1978	\fontspec_if_detect_external:n ... 561
\fontdimen 1243, 1263, 1267–1269, 1273–1275, 1280, 1281	\fontspec_if_detect_external:nTF 550, 561
\fontfamily 281, 300	\fontspec_if_feature:n 414
\fontname 730, 733	\fontspec_if_feature:nnn 433
\fontshape 1860, 1871	\fontspec_if_feature:nnnTF 1 , 433
\fontspec 240	\fontspec_if_feature:nTF 1 , 414
	\fontspec_if_fontspec_font: 377
	\fontspec_if_fontspec_font:TF .. 1 , 377
	\fontspec_if_language:n 461
	\fontspec_if_language:nn 477
	\fontspec_if_language:nnTF 1 , 477
	\fontspec_if_language:nTF 1 , 461
	\fontspec_if_opentype: 401
	\fontspec_if_opentype:TF 1 , 401
	\fontspec_if_script:n 448
	\fontspec_if_script:nTF 1 , 448
	\fontspec_info:n 69, 608, 1238, 2004
	\fontspec_info:nx 70, 735

\fontspec_info:nxx	71, 536	\gdef	33
\fontspec_init:	527, <u>832</u>	\global	62, 558, 579, 582
\fontspec_iv_str_to_num:n	\grave	1907
.....	438, 439, 483, <u>990</u> , 1015, 1034	\group_begin:	1232
\fontspec_iv_str_to_num:w	... 991, 993	\group_end:	1239
\fontspec_make_AAT_feature:nn	903, 914		
\fontspec_make_AAT_feature_string:nn 388, 874, 920, <u>966</u>	H	
\fontspec_make_feature:nnn	\hat	1913
.....	899, 947, 1523, 1583, 1590	\hbox	36
\fontspec_make_feature:nxx 1532	\hyphenchar	... 12, 15, 1291, 1300, 1302
\fontspec_make_ICU_feature:n		
.....	906, 910, 929	I	
\fontspec_namewrap:n	... 572, 856, 1086	\if@tempswa 670,
\fontspec_new_lang:nn	... 357, 358, 360	1026, 1030, 1046, 1052, 1070, 1079
\fontspec_new_script:nn	.. 340, 341, 343	\if@zf@configfile 39, 2009
\fontspec_patch_fancyvrb: 46, 68	\if@zf@math 40, 2003
\fontspec_patch_listings: 47, 73	\ifcase 704
\fontspec_patch_moreverb: 45, 58	\ifcsname 291,
\fontspec_patch_verbatim: 44, 49	316, 325, 378, 385, 402, 415, 434,
\fontspec_preparse_features:nn	531, <u>549</u>	449, 462, 478, 493, 507, 577, 578, 1867
\fontspec_save_family:n 576	\ifdefined 1968
\fontspec_save_family:nT	... 535, <u>576</u>	\iffontchar 19
\fontspec_save_fontinfo:nn	.. 537, <u>620</u>	\ifmmode 36
\fontspec_select:nn	278, 295, 522, <u>525</u> , 548	\ifnum 12, 708, 969,
\fontspec_set_bold: 540, <u>637</u>	1018, 1019, 1038, 1039, 1061, 1062
\fontspec_set_bold_italic:	.. 543, <u>665</u>	\ifodd 972
\fontspec_set_bold_slanted:	.. 544, <u>690</u>	\ifx 389,
\fontspec_set_family:Nnn	... <u>1</u> , 241,	639, 650, 660, 671–673, 691, 692,
.....	246, 250, 254, 259, 262, 265, 268, <u>521</u>	728, 734, 740, 743, 745, 751, 875,
\fontspec_set_font_dimen:NnN	921, 968, 978, 983, 997, 998, 1137,
.....	1234, 1235, <u>1241</u>	1148, 1171, 1262, 1521, 1530,
\fontspec_set_font_type:	.. 404, 417,	1581, 1866, 1919, 1981, 1991, 1995
.....	436, 451, 464, 480, 495, 509, 557, <u>701</u>	\ifzf@atsui 32, 387, 715, 873, 902
\fontspec_set_italic: 541, <u>648</u>	\ifzf@firsttime 27,
\fontspec_set_scriptlang:	... 532, <u>595</u>	816, 895, 1210, 1256, 1366, 1389, 1402
\fontspec_set_slanted: 542, <u>659</u>	\ifzf@graphite 35
\fontspec_set_upright: 539, <u>633</u>	\ifzf@icu 33, 405, 418,
\fontspec_setkeys:xx	56, 552, 554, 771, 824	437, 452, 465, 481, 496, 510, 596,
\fontspec_setkeys:xxx	717, 799, 879, 905, 1356, 1364, 1589
.....	60, 604, 605, 615, 616	\ifzf@math@euler 36, 1923
\fontspec_trace:n 72	\ifzf@math@lucida 37, 1928
\fontspec_v_str_to_num:n	... <u>990</u> , 1058	\ifzf@mm 34, 1360, 1365
\fontspec_warning:n 66,	\ifzf@nobf 28, 638, 667
.....	209, 302, 917, 932, 1335, 1339, 1367	\ifzf@noit 29, 649, 668
\fontspec_warning:nx 67, 351,	\ifzf@nosc 30, 741
.....	368, 922, 939, 959, 1108, 1112, 1844	\ifzf@package@euler@loaded	.. 38, 1895
\fontspec_warning:nxx 68, 320, 329	\ifzf@tfm 31
		\ignorespaces 243, 304
G		\input 2
\g@addto@macro 236	\InputIfFileExists 2010
\Gamma 1939	\int_new:N 45
		\int_set:Nn 419, 421, 467, 994

<code>\int_use:N</code>	627, 629	<code>\l_tmpa_tl</code>	1099, 1105, 1113
<code>\intexpr_compare:nTF</code>	1102	<code>\Lambda</code>	1942
<code>\itdefault</code>	5, 652, 655, 675, 678, 682, 686, 745, 751, 789, 793, 1875, 1883, 1963, 1971, 1974	<code>\latinencoding</code>	235, 238
<code>\itshape</code>	8, <u>1873</u>	<code>\leavevmode</code>	36
K			
<code>\key@ifundefined</code>	319, 328, 954, 956, 958	<code>\let</code>	8, 9, 33, 38, 70, 75, 289, 293, 306, 529, 530, 553, 555, 738, 821, 823, 870, 986, 1892, 1893, 1918, 1978
<code>\keyval@alias@key</code>	336, <u>949</u> , 960, 962, 964, 1319	<code>\listing@line</code>	62
L			
<code>\l_fontspec_extension_tl</code>	572, 1093, 1098	<code>listingcont*</code> (environment)	58
<code>\l_fontspec_extensions_clist</code>	374, 563	<code>\loop</code>	1018, 1038, 1061
<code>\l_fontspec_external_bool</code>	669, 1081, 1085, 1094	<code>\lst@visiblespace</code>	75
<code>\l_fontspec_hexcol_tl</code>	828, 829, 1313, 1316	<code>\luatex_if_engine:T</code>	18, 24, 861, 1467
<code>\l_fontspec_hyphenchar_tl</code>	1295–1297, 1300, 1303	<code>\luatex_if_engine:TF</code>	9
<code>\l_fontspec_lang_name_tl</code>	155, 601, 602, 605, 612, 613, 616, 849, 1128	<code>\luatexpstexhyphenchar</code>	866
<code>\l_fontspec_lang_tl</code>	365, 424, 631, 803, 812, 850, 1076, 1837, 1842, 1850	<code>\luatexpsthyphenchar</code>	864
<code>\l_fontspec_mode_tl</code>	808, 862, 1113	<code>\luatexpreehyphenchar</code>	865
<code>\l_fontspec_nfss_tl</code>	760, 767, 774, 787	<code>\luatexprehyphenchar</code>	863, 1303
<code>\l_fontspec_opacity_tl</code>	829, 852, 1324	<code>\luatexRequireModule</code>	11
<code>\l_fontspec_optical_size_tl</code>	574, 859, 1357, 1372	M	
<code>\l_fontspec_pre_feat_tl</code>	624, 763, 777, <u>797</u>	<code>\mathalpha</code>	1906–1915, 1929–1949
<code>\l_fontspec_rawfeatures_sclist</code>	624, 763, 777, 783, 821, 896	<code>\mathbf</code>	105, 1964, 1970
<code>\l_fontspec_renderer_tl</code>	573, 714, 716, 718, 860, 1105	<code>\mathbin</code>	1950
<code>\l_fontspec_scale_tl</code>	198, 761, 775, 822, 1227–1229, 1237	<code>\mathchardef</code>	1917
<code>\l_fontspec_script_name_tl</code>	155, 165, 597, 600, 604, 615, 847, 1123	<code>\mathclose</code>	1924, 1927, 1953, 1955
<code>\l_fontspec_script_tl</code>	348, 423, 466, 482, 630, 800, 802, 809, 811, 848, 1050, 1076	<code>\mathdollar</code>	1957
<code>\l_fontspec_size_tl</code>	769, 772, 775, 1217	<code>\mathit</code>	105, 1874, 1963, 1971, 1974
<code>\l_fontspec_sizedfont_tl</code>	770, 776, 1219	<code>\mathopen</code>	1952, 1954
<code>\l_fontspec_sizefeat_clist</code>	757, 768, 851, 1214	<code>\mathord</code>	1956, 1957
<code>\l_fontspec_strnum_int</code>	45, 349, 366, 438, 439, 483, 994, 1019, 1039, 1063, 1835, 1840	<code>\mathpunct</code>	1920, 1926
<code>\l_fontspec_tmpa_clist</code>	1519, 1525, 1528, 1534, 1579, 1585	<code>\mathrel</code>	1925, 1951
<code>\l_tmpa_bool</code>	565, 567, 569	<code>\mathring</code>	1915
<code>\l_tmpa_num</code>	1099, 1102	<code>\mathrm</code>	1962, 1969, 1973
		<code>\mathsf</code>	1965, 1976
		<code>\mathtt</code>	1966, 1977
		<code>\mddefault</code>	635, 652, 655, 662, 1960–1963, 1965, 1966, 1969, 1971
		<code>\MessageBreak</code>	783
		<code>\msg_error:nn</code>	13, 64
		<code>\msg_error:nnx</code>	65
		<code>\msg_info:nn</code>	69
		<code>\msg_info:nnx</code>	70
		<code>\msg_info:nnxx</code>	71
		<code>\msg_new:nnn</code>	4, 73, 85, 123, 128, 132, 136, 141, 146, 151, 157, 161, 167, 171, 175, 179, 184, 188, 192, 196, 200, 204
		<code>\msg_new:nnnn</code>	78, 89, 97, 107, 115
		<code>\msg_redirect_module:nnn</code>	216, 217, 220, 221
		<code>\msg_redirect_name:nnn</code>	1336
		<code>\msg_trace:nn</code>	72

\msg_warning:nn	66	\providecommand	1857
\msg_warning:nxx	67	\Psi	1948
\msg_warning:nxxx	68		
\multi@alias@key	334, <u>953</u>	Q	
		\q_nil	991, 993
N		R	
\newAATfeature	<u>315</u>	\real	1323
\newcommand	725, 815, 853, 869, 894, 943, 946, 949, 953, 1863	\relax	36, 349, 366, 438, 439, 483, 976, 1835, 1840, 1851, 1859, 1878, 1882, 1886, 1917
\newcount	41–44, 582	\repeat	1025, 1045, 1069
\newcounter	1320	\RequirePackage ...	1, 2, 10, 225–227, 232
\newfontface	<u>277</u>	\rmdefault	229, 246, 274
\newfontfamily	<u>277</u>	\rmfamily	1233, 1246
\newfontfeature	<u>307</u>		
\newfontlanguage	<u>355</u> , 1639–1832	S	
\newfontscript	<u>338</u> , 1598–1638	\scan_stop:	50, 1300, 1303
\newICUfeature	<u>324</u>	\scdefault	745, 751, 1875, 1879, 1883, 1887
\newif	27–40	\sclist_put_right:Nn	<u>887</u>
\newopentypefeature	<u>324</u>	\scshape	<u>1873</u>
\next	1918	\selectfont ...	242, 281, 300, 1860, 1871
\nhex	1325	\setboldmathrm	<u>258</u>
\normalfont	247, 251, 255	\setcounter	1323
\not@math@alphabet 1859, 1874, 1878, 1882, 1886	\setkeys	58, 62, 551, 944, 1089, 1095, 1122, 1127, 1337, 1520, 1529, 1580
\null	36	\setlength	1263, 1267–1269, 1280
\numexpr	976	\setmainfont	<u>245</u> , 257
O		\SetMathAlphabet	1962–1966, 1969–1971, 1973, 1974, 1976, 1977
\Omega	1949	\setmathrm	<u>258</u>
\or	706, 711	\setmathsf	<u>258</u>
P		\setmathtt	<u>258</u>
\PackageInfo	781	\setmonofont	<u>245</u>
\par	63	\setromanfont	<u>257</u>
\Phi	1947	\setsansfont	<u>245</u>
\Pi	1944	\SetSymbolFont	1905, 1961, 1967
\prg_case_int:nnn	1247	\settoheight	1245
\prg_case_str:nnn	1222	\sfdefault	230, 250, 275
\prg_new_conditional:Nnn	49, 377, 384, 401, 414, 433, 448, 461, 477, 492, 506, 561, 576, 1013, 1032, 1055	\sidefont	745, 751, 1857, 1860, 1875, 1879, 1883, 1887
\prg_return_false:	53, 381, 390, 395, 398, 408, 411, 425, 427, 430, 440, 442, 445, 453, 455, 458, 469, 471, 474, 484, 486, 489, 498, 500, 503, 512, 514, 517, 569, 593, 1026, 1030, 1046, 1052, 1070, 1079	\Sigma	1945
\prg_return_true: 51, 379, 392, 406, 425, 440, 453, 469, 484, 498, 512, 569, 593, 1026, 1030, 1046, 1052, 1070, 1079	\sishape	<u>1857</u>
\ProcessOptions	224	\sldefault	662, 694, 698, 792, 1879
		\slshape	1877, 1878
		\space	92, 198, 206, 970, 973, 976
		\stepcounter	308
		\string	105, 125
		\strip@pt	1237
		T	
		\textsi	<u>1857</u>

\textvisiblespace	20, 28	\use_i:nn	17
\the	63, 309, 588, 1273–1275, 1281	\use_ii:nn	19
\thelisting@line	62	\use_iv:nnnnn	48
\Theta	1941	\use_none:n	1009
\tilde	1909	\use_v:nnnnn	47
\tl_clear:N		\usefont	27
767, 769, 822, 832–850, 859, 860, 1098		\UTFencname	233
\tl_gset:cx	621–623, 626, 628		
\tl_gset:Nx	1237	V	
\tl_gset_eq:cN	630, 631	\value	1325
\tl_if_empty:NF	800, 809, 825, 1394, 1407	\verb	35
\tl_if_empty:NT	601, 612, 714, 772	\verb*	35
\tl_if_empty:NTF	597, 888	\verb@eol@error	38
\tl_if_empty:nTF	915, 930	verbatim* (environment)	49
\tl_if_eq:nnTF	520, 1246, 1289, 1334	\verbatim@font	39
\tl_if_eq:nvTF	497, 511	\verbatim@line	63
\tl_if_eq:xxT	789	\verbatim@processline	61
\tl_if_eq:xxTF	5	\verbatim@start	52, 65
\tl_if_head_eq_charcode_p:nN	1004, 1005		
\tl_if_in:nnT	566	X	
\tl_if_single:nTF	1294	\xdef	591, 817, 896
\tl_put_right:Nn		\xetex_if_engine:F	8
891, 1291, 1299, 1391, 1404		\xetex_if_engine:T	1089, 1122, 1127
\tl_put_right:Nx	774, 1395, 1408	\xetex_if_engine:TF	16, 21
\tl_replace_all_in:Nnn	1184	\xetex_or luatex:nn	
\tl_replace_all_in:Nnx	1182	16, 871, 900, 1103, 1111, 1355, 1440	
\tl_set:cn	1116–1120	\xetex_or luatex:nnn	
\tl_set:Nn	228–231, 348, 365, 482, 600,	21, 228, 701, 797, 856,	
602, 613, 716, 718, 797, 852, 862,		1013, 1032, 1054, 1086, 1299, 1328	
889, 1093, 1123, 1128, 1181, 1214,		\XeTeXcountvariations	708
1217, 1295, 1296, 1316, 1357,		\XeTeXfeaturename	967
1372, 1390, 1403, 1837, 1842, 1850		\XeTeXfonttype	704
\tl_set:Nv	423, 424, 466, 1105, 1113	\XeTeXisexclusivefeature	969
\tl_set:Nx	374, 760, 1227, 1229, 1324	\XeTeXOTcountfeatures	1057
\tl_set_eq:NN	233–235, 237,	\XeTeXOTcountlanguages	1035
238, 523, 770, 1392, 1397, 1405, 1410		\XeTeXOTcountscripts	1016
\token_to_str:N	1311	\XeTeXOTfeaturetag	1062
\ttdefault	231, 254, 276	\XeTeXOTlanguagetag	1039
\two@digits	1532	\XeTeXOTscripttag	1019
\typeout	2011, 2012	\XeTeXselectorname	970, 973, 976
		\Xi	1943
U		\XKV@rm	554, 555,
\unless	316, 325, 577, 638, 649,	773, 825, 826, 1519, 1521, 1525,	
728, 741, 743, 875, 895, 968, 972,		1528, 1530, 1534, 1579, 1581, 1585	
978, 983, 1210, 1256, 1521, 1530, 1581		\XKV@tfam	
\updefault	635,	138, 143, 148, 153, 1522, 1531, 1582	
641, 644, 1887, 1960–1962, 1964–		\XKV@tkey	138, 143, 148, 153
1967, 1969, 1970, 1973, 1976, 1977		\xllx@defaultthyphenchar	13, 17
\upshape	9, 1873	\xxt@visiblespace	18, 33, 70, 75
\Upsilon	1946	\xxt@visiblespace@fallback	22, 25
\use:c	386, 420, 422, 468	\xxt@vprintspaces	31, 42, 52, 55, 65
\use:x	46, 58, 62, 279, 294, 785, 791		

Z

\ze	12, 1017, 1036, 1059	\zf@fontname	87, 134, 139, 144, 149, 154, 159, 164, 198, 206, 528–530, 553, 624, 634, 640, 651, 674, 737, 749, 762, 770, 1182, 1392, 1397, 1405, 1410
\zap@space	374, 587, 1177, 1211, 1215	\zf@fontspec	<u>548</u>
\zf@	1257, 1260	\zf@graphitfalse	703
\zf@iii	1257	\zf@icufalse	703, 854
\zf@iiii	1257	\zf@icutrue	712, 723
\zf@adjust	787, 793, 823, 1271, 1272, 1281, 1291, 1299	\zf@it .	650, 654, 673, 677, 833, 1153, 1392
\zf@atsuifalse	703	\zf@it@feat	652, 655, 842, 1194, 1391
\zf@atsuitrue	707	\zf@make@font@shapes	
\zf@basefont	634, 640, 643, 651, 654, 661, 674, 677, 681, 685, 693, 697, <u>725</u>
386, 403, 416, 435, 450, 463, 479, 494, 508, 556, 558, 559, 704, 708, 738, 967, 969, 970, 973, 976, 1016, 1019, 1035, 1039, 1057, 1062, 1235, 1263, 1267–1269, 1280, 1297		\zf@make@smallcaps	742, 869
\zf@bf .	639, 643, 672, 681, 832, 1142, 1405	\zf@math	<u>1889</u> , 2005
\zf@bf@feat	641, 644, 841, 1190, 1404	\zf@math@eulertrue	1896
\zf@bfit .	671, 685, 836, 1158, 1397, 1410	\zf@math@lucidatrue	1901–1903
\zf@bfit@feat	675, 678, 682, 686, 843, 1198, 1395, 1408	\zf@math@maybe	<u>1979</u>
\zf@bfsl	691, 697, 838, 1166	\zf@merge@shape	
\zf@bfsl@feat	694, 698, 845, 1206	<u>1863</u> , 1875, 1879, 1883, 1887
\zf@default@options		\zf@mmfalse	703
.....	190, 288, 289, 293, 552, 622	\zf@mmtrue	709
\zf@define@feature@option		\zf@nobffalse	1141
322, 331, <u>943</u> , 1415–1438, 1449– 1457, 1459–1466, 1468, 1471– 1484, 1486–1496, 1498–1502, 1504, 1506–1512, 1514–1517, 1538–1548, 1550–1556, 1558–1577		\zf@nobftrue	1083, 1138
\zf@define@font@feature		\zf@noitfalse	1152
.....	317, 326, <u>943</u> , 1414, 1448, 1458, 1470, 1485, 1497, 1505, 1513, 1537, 1549, 1557, 1587	\zf@noittrue	1084, 1149
\zf@enc	228, 232–235, 237, 238, 538, 786, 792, 1960–1967, 1969–1971, 1973, 1974, 1976, 1977	\zf@noscfalse	1175
\zf@fake@embolden .	835, 1394, 1396, 1403	\zf@nosctrue	1172
\zf@fake@slant ...	834, 1390, 1407, 1409	\zf@package@euler@loadedfalse ...	1890
\zf@family		\zf@package@euler@loadedtrue	1889
281, 300, 420, 422–424, 466, 468, 497, 511, 523, 538, 591, 592, 621– 623, 626, 628, 630, 631, 786, 792, 793		\zf@rmboldmaths	262, 1968–1971
\zf@family@long ..	529, 577, 586, 591, 817	\zf@rmmaths	
\zf@firsttimefalse	534	259, 274, 1960–1964, 1967, 1973, 1974	
\zf@firsttimetrue	855	\zf@esc	740, 749, 839, 1176
\zf@font@feat		\zf@esc@feat	745, 751, 846, 1210
... 533, 555, 635, 641, 644, 652, 655, 662, 675, 678, 682, 686, 694, 698		\zf@sfmaths	265, 275, 1965, 1976
\zf@font@warning	1892, 1978	\zf@sl	660, 661, 692, 693, 837, 1162
		\zf@sl@feat	662, 844, 1202
		\zf@smallcaps	743, 744, 870, 876, 880, 884
		\zf@tfmfalse	703
		\zf@tfmtrue	705
		\zf@ttmaths	268, 276, 1966, 1977
		\zf@up	530, 553, 556, 558, 1132
		\zf@up@feat	635, 840, 1186
		\zf@update@family	311, 347, 364, <u>815</u> , 924, 936, 1101, 1124, 1129, 1133, 1139, 1143, 1150, 1154, 1159, 1163, 1167, 1173, 1177, 1187, 1191, 1195, 1199, 1203, 1207, 1211, 1215, 1228, 1255, 1279, 1284, 1288, 1310,

1322, 1331, 1343, 1347, 1351,	<u>894</u> , 925, 937, 1285, 1332, 1344,
1358, 1361, 1373, 1377, 1381,	1348, 1352, 1362, 1378, 1382,
1385, 1393, 1406, 1441, 1444,	1386, 1442, 1445, 1592, 1595, 1855
1591, 1594, 1836, 1841, 1849, 1854	
\zf@update@ff 312, 829,	\zf@wordspace@parse 1257, <u>1260</u>